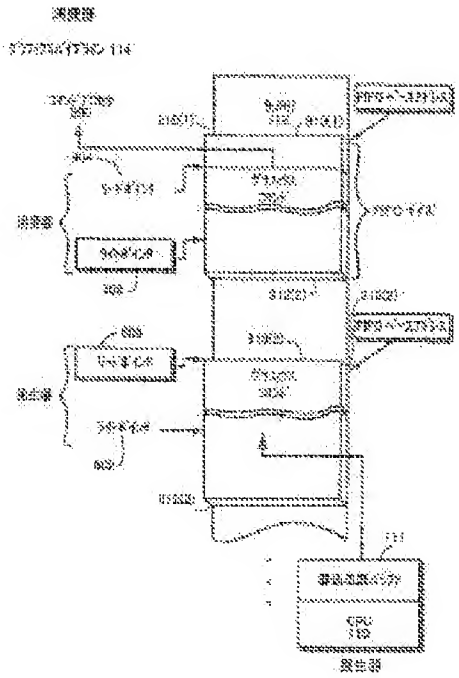
GRAPHICS SYSTEM

Publication number: JP2002132489 (A)
Publication date: 2002-05-10
Inventor(s): FOULADI FARHAD; MOORE ROBERT +
Applicant(s): NINTENDO CO LTD +
Classification:
- international: ***G06F3/153; G06F5/06; G06T1/60; G06T15/00;*** **G06F3/153; G06F5/06; G06T1/60; G06T15/00;** (IPC1-7): G06F3/153; G06F5/06; G06T15/00
- European: G06F3/14; G06T1/60
Application number: JP20010195710 20010628
Priority number(s): US20000226912P 20000823; US20000726215 20001128

Also published as:
US7196710 (B1)
US2007165043 (A1)
US7701461 (B2)

Abstract of **JP 2002132489 (A)**

PROBLEM TO BE SOLVED: To provide a technique effectively buffering a graphics command between a graphics command generating device and a graphics command consuming device. SOLUTION: A graphics system is provided with a 3D graphics pipeline and a graphics/audio processor including an audio DSP. A variable and a graphics command buffer 812 are arranged by allocating a part of a main memory 112. All writing into a graphics FIFO is fed to a buffer in the main memory 112. The generating device and consuming device independently maintains their own leads and writing pointers 802 and 808. The consuming device uses the writing pointer 808 in order to track an available position of data in the buffer. The generating device reads a series of graphics commands stored in a certain place of the memory to the buffer and then write a lead command so as to read the remainder of the buffer.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－132489
（P2002－132489A）
(43)公開日　平成14年５月10日(2002.5.10)

(51)Int.Cl.[7]	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　3/153	３３６	Ｇ０６Ｆ　3/153	３３６Ａ	５Ｂ０６９
5/06		5/06	Ｚ	５Ｂ０８０
Ｇ０６Ｔ　15/00	１００	Ｇ０６Ｔ　15/00	１００Ａ	

審査請求　未請求　請求項の数81　ＯＬ　外国語出願　（全101頁）

(21)出願番号　特願2001－195710(P2001－195710)

(22)出願日　平成13年６月28日(2001.6.28)

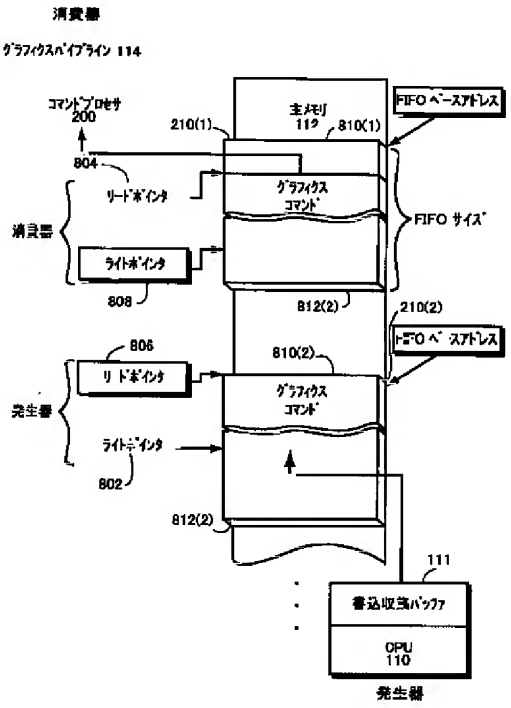
(31)優先権主張番号　６０／２２６９１２
(32)優先日　平成12年８月23日(2000.8.23)
(33)優先権主張国　米国（ＵＳ）
(31)優先権主張番号　０９／７２６２１５
(32)優先日　平成12年11月28日(2000.11.28)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　000233778
任天堂株式会社
京都府京都市南区上鳥羽鉾立町11番地１
(72)発明者　ファーハッド　フォーラディ
アメリカ合衆国　カリフォルニア州　パロアルト　アルマ通り　101
(72)発明者　ロバート　ムーア
アメリカ合衆国　ワシントン州　レドモンド　227番プレース　エヌイー　2522
(74)代理人　100090181
弁理士　山田　義人

Ｆターム(参考)　5B069 AA16
5B080 CA04 CA08 CA09

(54)【発明の名称】　グラフィクスシステム

(57)【要約】　（修正有）
【課題】グラフィクスコマンド発生器とグラフィクスコマンド消費器との間でグラフィクスコマンドを効率的にバッファリングする技術を提供する。
【解決手段】グラフィクスシステムは３ＤグラフィクスパイプラインおよびオーディオＤＳＰを含むグラフィクス／オーディオプロセサを備える。主メモリ１１２の一部を割り付けることによって、可変数、可変サイズのグラフィクスコマンドバッファ８１２を設ける。グラフィクスＦＩＦＯへの全ての書き込みが主メモリ１１２中のバッファに送られる。発生器および消費器は独立して自身のリードおよびライトポインタ８０２，８０８を維持する。消費器はバッファ内のデータの有効位置をトラッキングするためにライトポインタ８０８を使用する。発生器は、バッファへ、消費器がメモリのどこかへストアされている一連のグラフィクスコマンドを読み出しかつその後バッファの残りを読み出すようにリードコマンドを書き込む。



【特許請求の範囲】
【請求項１】グラフィクスコマンドを出力する発生器、発生器によって出力されたグラフィクスコマンドを消費する消費器、および発生器および消費器の間に結合され、消費器への分配のために、発生器によって出力されたグラフィクスコマンドを受けかつ一時的にストアする少なくとも１つのバッファをストアする記憶装置を備え、
発生器および消費器は前記バッファを他方とは独立してアクセスすることができる、グラフィクスシステム。
【請求項２】発生器および消費器は独立したリードおよび／またはライトポインタを有する、請求項１記載のグラフィクスシステム。
【請求項３】記憶装置はその記憶装置内の選択された位置に配置された複数のサイズ可変バッファをストアする、請求項１記載のグラフィクスシステム。
【請求項４】記憶装置は各々が発生器および／または消費器によって独立してアクセスされ得る複数のバッファをストアする、請求項１記載のグラフィクスシステム。
【請求項５】消費器は、複数のバッファの少なくともアクティブな１つへ書き込みすることができないが、その複数のバッファの少なくとも前記アクティブな１つのためのライトポインタを発生器とは独立して維持する、請求項４記載のグラフィクスシステム。
【請求項６】消費器は発生器がアクティブなバッファへ書き込んだことに応答して消費器ライトポインタを選択的にインクリメントする、請求項２記載のグラフィクスシステム。
【請求項７】発生器は前記複数のバッファの第１のバッファに関連する発生器リードポインタおよび発生器ライトポインタを与え、消費器は前記複数のバッファの前記第１バッファに関連する消費器リードポインタおよび消費器ライトポインタを独立して維持する、請求項４記載のグラフィクスシステム。
【請求項８】消費器は、その消費器がアクティブなバッファから読み出すことに応じて消費器リードポインタをインクリメントし、そのインクリメントした消費器リードポインタが消費器ライトポインタと所定の関係を有するとき、アクティブなバッファからの読出を中断する、請求項６記載のグラフィクスシステム。
【請求項９】複数のバッファの第１バッファは、発生器が記憶装置内のどこかへストアしたグラフィクスコマンドのセットを消費させかつどこかにストアされたそのグラフィクスコマンドを消費した後にはその第１バッファからのグラフィクスコマンドの消費を再開させるように、消費器を制御するリードコマンドを含む、請求項４記載のグラフィクスシステム。
【請求項１０】リードコマンドは、表示リストの開始アドレスと長さとを特定し、その特定した開始アドレスで始まる特定した長さの表示リストを読み出すように消費器を制御する、請求項９記載のグラフィクスシステム。
【請求項１１】バッファは循環的な先入れ先出し（ＦＩＦＯ）アクセスを与える、請求項１記載のグラフィクスシステム。
【請求項１２】バッファは直線的な先入れ先出し（ＦＩＦＯ）アクセスを与える、請求項１記載のグラフィクスシステム。
【請求項１３】バッファは発生器および消費器の両方へ同時に選択的に付属され得る、請求項１記載のグラフィクスシステム。
【請求項１４】バッファは発生器に付属され、別のバッファが消費器に付属される、請求項１記載のグラフィクスシステム。
【請求項１５】バッファはまず発生器に付属され、次いで、発生器から離されかつ消費器に付属される、請求項１記載のグラフィクスシステム。
【請求項１６】バッファは発生器、消費器、またはその両方へ付属され得る、請求項１記載のグラフィクスシステム。
【請求項１７】１つのバッファだけが一度に発生器に付属される、請求項１記載のグラフィクスシステム。
【請求項１８】１つのバッファだけが一度に消費器に付属される、請求項１記載のグラフィクスシステム。
【請求項１９】バッファは１６Ｋバイトの最大サイズを有する、請求項１記載のグラフィクスシステム。
【請求項２０】発生器はバッファのサイズをセットする、請求項１記載のグラフィクスシステム。
【請求項２１】バッファはグラフィクスコマンドのフレームをストアするために動的にサイズを決められる、請求項１記載のグラフィクスシステム。
【請求項２２】発生器は、バッファ開始アドレスおよびバッファ長を特定するグラフィクスバッファ初期化コマンドを発行することによってバッファを宣言する、請求項１記載のグラフィクスシステム。
【請求項２３】バッファは３２バイトの倍数である長さを有しかつ６４Ｋバイトの最小サイズを有する、請求項１記載のグラフィクスシステム。
【請求項２４】発生器はバッファに中断点を書き込み、消費器はその中断点への遭遇に応じてグラフィクスコマンドの消費を中断する、請求項１記載のグラフィクスシステム。
【請求項２５】バッファは、発生器がバッファ内の位置に上書きしたときを示すオーバフローステータス表示器を有する、請求項１記載のグラフィクスシステム。
【請求項２６】さらに、バッファのステータスを示すハードウェアステータスレジスタを備える、請求項１記載のグラフィクスシステム。
【請求項２７】ステータスレジスタは、バッファフルおよびバッファエンプティに関連する発生器ライトポインタの位置；バッファオーバフロー；発生器が現在バッフ

ァへ書き込んでいるかどうか；および消費器が現在バッファから読み出しているかどうかのパラメータを含む、請求項２６記載のグラフィクスシステム。

【請求項２８】さらに、バッファに結合され、書込が読出を追い越すのを防止するために、フロー制御ロジックを与えるハードウェアコントローラを備える、請求項１記載のグラフィクスシステム。

【請求項２９】さらに、バッファに結合され、リードおよびライトポインタを最後の位置から最初の位置へラップするハードウェアコントローラを備える、請求項１記載のグラフィクスシステム。

【請求項３０】発生器はプロセサを含み、消費器はグラフィクスパイプラインを有するグラフィクスプロセサを含む、請求項１記載のグラフィクスシステム。

【請求項３１】記憶装置は主システムメモリを含み、発生器は主システムメモリ内において前記バッファを動的に割り付ける、請求項１記載のグラフィクスシステム。

【請求項３２】グラフィクスコマンドを受けかつ一時的にストアする記憶バッファ、

前記バッファ内へグラフィクスコマンドを書き込み、バッファに関連する発生器ライトポインタおよび発生器リードポインタを維持する発生器、および発生器ライトポインタとは独立した消費器ライトポインタおよび発生器リードポインタとは独立した消費器リードポインタを維持し、バッファ内にストアされているグラフィクスコマンドを消費する消費器を備える、グラフィクスシステム。

【請求項３３】消費器は消費器がバッファから消費するたび毎に消費器リードポインタをインクリメントし、消費器リードポインタが消費器ライトポインタと所定の関係を有するときバッファからの消費を中断する、請求項３２記載のグラフィクスシステム。

【請求項３４】消費器は発生器がバッファへ書込を行ったことに応答して消費器ライトポインタを選択的に自動的にインクリメントする、請求項３２記載のグラフィクスシステム。

【請求項３５】発生器は、消費器が発生器のバッファへの書込に応答して消費器ライトポインタを自動的にインクリメントすべきかどうかを特定する構成コマンドを消費器に送る、請求項３２記載のグラフィクスシステム。

【請求項３６】発生器ライトポインタに基づいてバッファ中へグラフィクスコマンドを書き込むグラフィクスコマンド発生器と、消費器リードポインタに基づいてバッファからグラフィクスコマンドを読み出すグラフィクスコマンド消費器とを含むグラフィクスシステムにおいて、

消費器によって消費器ライトポインタが独立的に維持され、その消費器ライトポインタは発生器が前記バッファ中へ書き込んだ有効データの範囲を示し、消費器は消費器リードポインタが消費器ライトポインタに対して所定の関係を有することに応答してバッファからのグラフィクスコマンドを消費することを中止するようにしたことを特徴とする、グラフィクスシステム。

【請求項３７】アプリケーションを実行するプロセサモジュール、グラフィクスプロセサモジュール、およびプロセサモジュールおよびグラフィクスプロセサモジュールに結合された少なくとも１つのメモリを含むインタラクティブなグラフィクスシステムにおいて、プロセサモジュールとグラフィクスプロセサモジュールとの間のグラフィクスコマンドのフローを制御する方法であって、アプリケーションの制御の下で、メモリ中の可変数のＦＩＦＯバッファを動的に確立し、そのアプリケーションがＦＩＦＯバッファの各々のサイズを特定し、アプリケーションは、複数のＦＩＦＯバッファの少なくとも第１のＦＩＦＯバッファ中にグラフィクスコマンドを書き込むようにプロセサモジュールを制御し、そしてアプリケーションは、グラフィクスプロセサモジュールが前記プロセサの書込とは独立して第１のＦＩＦＯバッファからグラフィクスコマンドを読み出すのを制御するグラフィクスコマンドをグラフィクスプロセサモジュールに送る、方法。

【請求項３８】プロセサモジュールは複数の第１のバッファに関連するプロセサモジュールリードポインタおよびプロセサモジュールライトポインタを与え、そしてグラフィクスプロセサモジュールは前記第１のバッファに関連するグラフィクスプロセサモジュールリードポインタおよびグラフィクスプロセサモジュールライトポインタを独立して維持する、請求項３７記載の方法。

【請求項３９】グラフィクスプロセサモジュールはグラフィクスプロセサモジュールが第１のバッファから読み出すたび毎にグラフィクスプロセサモジュールリードポインタをインクリメントし、グラフィクスプロセサモジュールリードポインタがグラフィクスプロセサモジュールライトポインタと所定の関係を有するとき第１のバッファからの読出を中断する、請求項３７記載の方法。

【請求項４０】グラフィクスプロセサモジュールは、プロセサモジュールが第１のバッファへ書き込むことに応答してグラフィクスプロセサモジュールライトポインタを選択的に自動的にインクリメントする、請求項３７記載の方法。

【請求項４１】グラフィクスプロセサモジュールは、プロセサモジュールとは独立して、複数のバッファの少なくともアクティブな１つのためのライトポインタを維持する、請求項３７記載の方法。

【請求項４２】さらに、中断点を設定し、そしてグラフィクスプロセサモジュールがその中断点に遭遇することに応答してグラフィクスプロセサモジュールがバッファを読み出すのを少なくとも一時的に中断するようにした、請求項３７記載の方法。

【請求項４３】選択的な制御ステップは、オーバフロー

の検出に応答してプロセサモジュールがバッファへ書き込むのを中断させるステップを含む、請求項３７記載の方法。
【請求項４４】グラフィクスデータのフローを制御する方法であって、
グラフィクスデータを各々が複数の記憶位置を有する複数のサイズ可変ＦＩＦＯバッファへ書き込むステップ、
複数の記憶位置の少なくとも１つに関連する中断点を設定するステップ、
所定の順序で複数のバッファからグラフィクスデータを読み出すステップ、
中断点に関連する少なくとも１つの記憶位置への遭遇に応答して読出ステップを一時的に中断して割り込みを発生するステップ、および割り込みクリアコマンドの受信に応答して読出ステップを再開するステップを含む、方法。
【請求項４５】再開ステップは中断点ディスエーブルコマンドの受信に応答して行われる、請求項４４記載の方法。
【請求項４６】書込ステップは、第１イメージフレームに対応するグラフィクスデータをバッファの第１部分へ書き込むステップ、および第２イメージフレームに対応するグラフィクスデータをバッファの第２部分へ書き込むステップを含み、そして設定ステップは第１および第２バッファ部分を分離する位置に関連して中断点を設定するステップを含む、請求項４４記載の方法。
【請求項４７】所定順序は先入れ先出し（ＦＩＦＯ）である、請求項４４記載の方法。
【請求項４８】書込ステップは汎用処理ハードウェアによって行われ、読出ステップは特定目的グラフィクスハードウェアによって行われる、請求項４４記載の方法。
【請求項４９】読出ステップおよび書込ステップは実質的に同時に行われる、請求項４４記載の方法。
【請求項５０】書込ステップは読出ステップが開始する前に完了する、請求項４４記載の方法。
【請求項５１】再開された読出ステップは、別の中断点に遭遇するか、バッファがエンプティになるかあるいはグラフィクスデータによって表されるイメージフレームが異常終了するまで継続する、請求項４４記載の方法。
【請求項５２】さらに、バッファのステータスを表示するステップを含む、請求項４４記載の方法。
【請求項５３】ステータス表示ステップは、バッファフルおよびバッファエンプティに関連するライトポインタの位置；バッファオーバフロー；書込ステップがアクティブであること；コマンド処理がアイドル状態であるかどうか；および読出ステップがアクティブであることの、パラメータの少なくとも３つを表示する、請求項５２記載の方法。
【請求項５４】書込ステップはバッファを循環的に書き込むステップを含み、読出ステップはバッファを循環的に読み出すステップを含む、請求項４４記載の方法。
【請求項５５】グラフィクスコマンドを受けかつそれを一時的にストアする記憶装置、
前記記憶装置中のどこかにストアされているグラフィクスコマンドの第１セットおよびグラフィクスコマンドの第２セットを参照するコマンドを含むコマンドを前記記憶装置中のバッファへ書き込む発生器、およびバッファ中にストアされているグラフィクスコマンドの第１セットを消費し、参照コマンドに遭遇したことに応答して、それによって参照されたグラフィクスコマンドの第２セットを消費し、その後バッファからの付加的なコマンドを消費するためにバッファへリターンする消費器を備える、グラフィクスシステム。
【請求項５６】バッファは循環バッファである、請求項５５記載のグラフィクスシステム。
【請求項５７】参照コマンドは表示リストの開始アドレスを特定し、その参照コマンドはその特定された開始アドレスで開始する表示リストを読み出すように消費器を制御する、請求項５５記載のグラフィクスシステム。
【請求項５８】参照コマンドは消費器が消費すべきデータユニットの数を特定する、請求項５５記載のグラフィクスシステム。
【請求項５９】消費器はバッファへの書込はできないが、発生器とは独立してバッファのためのライトポインタを維持する、請求項５５記載のグラフィクスシステム。
【請求項６０】グラフィクスシステムにおいて、グラフィクスコマンドの発生器からのグラフィクスコマンドをグラフィクスコマンドの消費器へ送る方法であって、
発生器および消費器に結合されたメモリ中の可変位置に配置された複数のサイズ可変バッファを作り、
サイズ可変バッファへ発生器によって発生されたグラフィクスコマンドを一時的にストアし、
発生器とは独立してバッファをアクセスすることによってサイズ可変バッファからのグラフィクスコマンドを消費器によって消費し、そして消費されたグラフィクスコマンドの少なくとも一部に基づいてグラフィクスイメージの少なくとも一部を発生する、方法。
【請求項６１】消費器は複数のバッファの少なくともアクティブな１つへ書き込むことはできず、さらに、発生器とは独立して、消費器が複数のバッファの少なくとも前記アクティブな１つのためのライトポインタを維持する、請求項６０記載の方法。
【請求項６２】さらに、前記複数のバッファの第１バッファに関連する発生器リードポインタおよび発生器ライトポインタを発生器によって維持し、消費器によって、独立して、前記複数のバッファの前記第１バッファに関連する消費器リードポインタおよび消費器ライトポインタを維持する、請求項６０記載の方法。
【請求項６３】さらに、消費器によって、アクティブな

バッファから消費器が読み出すことに応じて消費器リードポインタをインクリメントし、そしてインクリメントした消費器リードポインタが消費器ライトポインタと所定の関係を有するときアクティブなバッファからの読出を中断する、請求項６２記載の方法。
【請求項６４】さらに、発生器がアクティブなバッファへ書き込むことに応答して選択的に消費器ライトポインタをインクリメントする、請求項６３記載の方法。
【請求項６５】複数のバッファの第１バッファはリードコマンドを含み、さらに、
(a) リードコマンドに遭遇することに応答して、発生器が記憶装置のどこかにストアしたグラフィクスコマンドのセットを消費し、
(b) どこかにストアされたグラフィクスコマンドを消費した後、第１バッファからのグラフィクスコマンドの消費を再開する、請求項６０記載の方法。
【請求項６６】リードコマンドは表示リストの開始アドレスおよび長さを特定し、そしてステップ(a) では特定した開始アドレスで開始する特定した長さの表示リストを読み出すように消費器を制御する、請求項６５記載の方法。
【請求項６７】複数のバッファの第１バッファは循環的な先入れ先出し（ＦＩＦＯ）アクセスを提供する、請求項６０記載の方法。
【請求項６８】複数のバッファの第１バッファは直線的な先入れ先出し（ＦＩＦＯ）アクセスを提供する、請求項６０記載の方法。
【請求項６９】さらに、発生器および消費器の両方へ複数のバッファのいずれかを同時に選択的に付属させる、請求項６０記載の方法。
【請求項７０】複数のバッファの第１バッファを発生器へ付属させ、複数のバッファの第２バッファを消費器へ付属させる、請求項６０記載の方法。
【請求項７１】複数のバッファの第１バッファを発生器へ付属させ、次いで、発生器から第１バッファを切り離し、第１バッファを消費器へ付属させる、請求項６０記載の方法。
【請求項７２】複数のバッファのいずれかを発生器へ、または消費器へ、もしくは両方へ付属させる、請求項６０記載の方法。
【請求項７３】複数のバッファの１つだけを一度に発生器へ付属させる、請求項６０記載の方法。
【請求項７４】複数のバッファの１つだけを一度に消費器へ付属させる、請求項６０記載の方法。
【請求項７５】イメージを発生する方法であって、
バッファに関連する発生器ライトポインタおよび発生器リードポインタを維持するステップ、
バッファへグラフィクスコマンドを書き込み、そしてその書込に応答して少なくともライトポインタを更新するステップ、
バッファに関連して、発生器ライトポインタとは独立した消費器ライトポインタおよび発生器リードポインタとは独立した消費器リードポインタを維持するステップ、
バッファ内にストアされたグラフィクスコマンドを消費し、その消費に応答して少なくともリードポインタを更新するステップ、および消費ステップに少なくとも部分的に応答してグラフィクスイメージの少なくとも一部を発生するステップを含む、方法。
【請求項７６】イメージを発生する方法であって、
記憶装置内のどこかにストアされたグラフィクスコマンドの第１セットおよびグラフィクスコマンドの第２セットを参照するコマンドを含むコマンドを前記記憶装置内のバッファへ書き込むステップ、
バッファにストアされているグラフィクスコマンドの第１セットを消費するステップ、
参照コマンドへの遭遇に応答して、グラフィクスコマンドの第２セットを消費し、その後、バッファからの付加的なコマンドを消費するように自動的にリターンするステップ、および消費されたグラフィクスコマンドの第１セットおよび第２セットに少なくとも部分的に応答してイメージの少なくとも一部を発生するステップを含む、方法。
【請求項７７】グラフィクスコマンドの第２セットは表示リストを含む、請求項７６記載の方法。
【請求項７８】データ構造子を発生する方法であって、
表示リストの開始を示す所定のコマンドを含むグラフィクスコマンドストリームをコマンドストリームバッファへ書き込むステップ、および所定のコマンドに応答して、コマンドストリームバッファからのグラフィクスコマンドストリームを表示リストバッファへ向け直すステップを含む、方法。
【請求項７９】コマンドストリームバッファは先入れ先出し（ＦＩＦＯ）バッファを含む、請求項７８記載の方法。
【請求項８０】さらに、表示リストバッファからのグラフィクスコマンドストリームをコマンドストリームバッファへ向け直す別の所定のコマンドを書き込むステップを含む、請求項７８記載の方法。
【請求項８１】３Ｄグラフィクスコマンドを３Ｄグラフィクスコマンド消費器へ供給する方法であって、
(a) 所定の記憶位置で開始するコマンドシーケンスをストアするステップ、および(b) ＦＩＦＯバッファを通してコマンド消費器を所定の記憶位置へ参照させる少なくとも１つのコマンドを含むグラフィクスコマンドストリームを消費器へ供給するステップを含み、
所定の記憶位置で開始するコマンドシーケンスを消費した後発生器はＦＩＦＯバッファへリターンする、方法。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】この発明はコンピュータグラフィ

クスに関し、より特定的には、家庭用ビデオゲームプラットフォームのようなインタラクティブなグラフィクスシステムに関する。さらに特定的には、この発明は、グラフィクスコマンド発生器とグラフィクスコマンド消費器との間での効率的なグラフィクスコマンドのバッファリングに関する。

【０００２】

【発明の背景および発明の概要】多くの人々はかなりリアルな恐竜，エイリアン，生き生きとしたおもちゃおよび他の空想的な動物を含む映画をかつて見たことがある。そのようなアニメーションはコンピュータグラフィクスによって可能とされた。そのような技術を用いて、コンピュータグラフィクスのアーティストは、各オブジェクトがどのように見えるべきかや時間の経過とともに外見上どのように変化すべきかを特定し、コンピュータは、そのオブジェクトをモデル化してテレビジョンやコンピュータスクリーンのようなディスプレイに表示する。コンピュータは、表示される映像の各部分を、場面中の各オブジェクトの位置や向き，各オブジェクトを照らすように見える照明の方向，各オブジェクトの表面テクスチャ，および他の要素に正確に基づいて、色付けしまた形作るために必要な多くのタスクを実行する。

【０００３】コンピュータグラフィクスの生成は複雑であるので、ここ数年前のコンピュータによって生成された３次元（３Ｄ）グラフィクスは、ほとんど高価な特殊なフライトシミュレータ，ハイエンドグラフィクスワークステーションおよびスーパーコンピュータに限られていた。大衆は映画や高価なテレビコマーシャルにおいてこれらのコンピュータシステムによって生成された映像のいくつかを見たが、大部分の人はグラフィクスを生成しているコンピュータに対して実際に相互作用をさせることはできない。たとえば、Nintendo64（登録商標）や今や利用可能であるパソコン用の種々の３Ｄグラフィクスカードのような比較的安価な３Ｄグラフィクスプラットフォームの利用によって、このすべてが変わった。今や、家庭や会社の比較的安価なコンピュータグラフィクスシステム上でエキサイティングな３Ｄアニメーションやシミュレーションに対して相互作用を及ぼすことができる。

【０００４】グラフィクスシステムの設計者が過去に直面した問題は、グラフィクスコマンド発生器とグラフィクスコマンド消費器との間でグラフィクスコマンドを如何に効率的にバッファリングするかということであった。この問題に対する種々のソリューションが提案されている。たとえば、グラフィクスコマンド発生器とグラフィクスコマンド消費器との間にバッファメモリを設けることがよく知られている。しばしば、このバッファメモリは、グラフィクスコマンド消費器の一部として（たとえば、グラフィクスチップのオンボードとして）接続される。グラフィクスコマンド発生器はグラフィクスコマンドをバッファメモリ中へ書き込み、そしてグラフィクスコマンド消費器はバッファメモリからこれらのグラフィクスコマンドを読み出す。そのようなバッファメモリのためには、典型的には、先入れ先出し（ＦＩＦＯ）バッファとして構成され、そのために、グラフィクスコマンド消費器は、グラフィクスコマンド発生器によってバッファ中へ書き込まれたと同じシーケンスでグラフィクスコマンドを読み出す。

【０００５】発生器と消費器との間にそのようなバッファを配置することは、発生器と消費器とが同期しなければならないという程度を緩和する。発生器は、消費器がバッファからコマンドを読み出すレートとは独立したレートでバッファ中へコマンドを書き込むことができる。たとえ消費器がバッファからの読出において一時的な遅れを受けたとしても（たとえば、発生器が消費器に大きなもしくは複雑なプリミティブを引き出させようとするときに生じるかもしれない）、発生器は、発生器がバッファを満杯にしない限り、そして新たなコマンドを書き込むためのメモリスペースがなくなるまで停止することはない。同じように、新たなグラフィクスコマンドをバッファ中へ書き込む際の発生器の一時的な遅れは、発生器が追加のグラフィクスコマンドを書き込む機会を有する前にバッファ中のすべてのグラフィクスコマンドを消費器が消費しない限り、消費器を停止させることはない。

【０００６】過去に遭遇した潜在的な問題は、バッファのサイズに関する。チップサイズや複雑さにおける制限のために、グラフィクスチップ上に非常に大きなコマンドバッファメモリを配置することはできない。グラフィクスハードウェア中の小さいサイズのＦＩＦＯバッファは、発生器と消費器との間の負荷バランスを適切にとれず、消費器が大きなプリミティブをレンダリングするときに発生器を停止させる。このように、過去に有意な仕事がなされたが、さらなる改良が可能である。

【０００７】この発明は、グラフィクスコマンド発生器とグラフィクスコマンド消費器との間でグラフィクスコマンドをより効率的にバッファリングする技術および構成を提供することによって、この問題を解決する。この発明の局面によれば、発生器と消費器との間で共用される主メモリの一部が可変数のサイズ可変グラフィクスコマンドバッファへ割り付けられる。発生器はバッファの数や各々のサイズを特定することができる。グラフィクス消費器への書込は主メモリ中のバッファの任意のものへ送られ得る。バッファは消費器および発生器へ同時に付属させられ(attached) 得て、もしくは異なるバッファが消費器および発生器へ付属させられてもよい。異なるバッファが消費器および発生器へ付属させられるこのマルチバッファの方法においては、発生器は１つのバッファへ書き込むことができ、他方消費器は他のバッファから読み出すことができる。

【０００８】消費器を発生器からさらに切り離すために、発生器および消費器は、この発明の他の局面に従って、それら自身のリードおよびライトポインタを独立的に維持する。消費器はバッファへ書き込むことはできないけれども、バッファ中の有効データの位置のトラッキングを保持するために使用するライトポインタを維持する。同じように、発生器はそれへ付属されていないバッファから読み出すことはできないが、バッファ中のデータの有効位置のトラッキングを保持するために使用するリードポインタを維持する。このポインタ構成の効果は、発生器を消費器からさらに切り離すことであり、それらの２つの間の同期化要求を減じる。

【０００９】この発明によって提供される他の局面によれば、発生器は、ＦＩＦＯバッファへ「表示リストコール」コマンドを書き込むことができ、このコマンドはメモリ中のどこかにストアされている一連のグラフィクスコマンド（たとえば、表示リスト）を消費器に読み出させ、そして続いて、バッファの残りの読出へリターンさせる。ＦＩＦＯバッファから異種の(out-of-line) グラフィクスコマンドストリングをコールするこの能力は、付加的な柔軟性を与え、そして同期化要求をさらに減じる。

【００１０】この発明の他の局面によれば、グラフィクスコマンド発生器は後続するコマンドを自動的に表示リストバッファへ向け直すコマンドを含むグラフィクスコマンドストリームを書き込むことができる。これを見えるようにする１つの方法は、グラフィクスコマンドのストリームを継続的に発生する向け直し可能な消火ホース(fire hose) としてグラフィクスコマンド発生器を想像することである。この消火ホースは通常グラフィクスコマンドをＦＩＦＯバッファへ流し込む。しかしながら、発生器は、このストリーム中に、「表示リスト開始」コマンドを含むことができ、このコマンドはそのコマンドに続くグラフィクスコマンドをそれに代えて表示リストへ書き込むようにする。ストリーム上にさらに挿入された「表示リスト終了」コマンドは、表示リストを終了させ、グラフィクスコマンドストリームを同じ（もしくは異なる）ＦＩＦＯバッファへ戻すように向け直すことができる。この特徴は、グラフィクスコマンド発生器によって非常に低いオーバヘッドで再使用可能な表示リストを効率的に生成することができるという利点を有する。

【００１１】この発明によって提供される他の局面によれば、グラフィクスコマンド発生器は、多数のＦＩＦＯバッファの任意のものへ中断点(break point) を挿入することができる。この中断点は消費器を中断させることができる。そのような中断点は、緊密な同期が必要なときに発生器と消費器とを同期化するのに役立つ。

【００１２】この発明によって提供されるさらに他の局面によれば、グラフィクスシステムは、グラフィクスコマンドを出力する発生器、発生器によって出力されたグラフィクスコマンドを消費する消費器、および発生器および消費器の間に結合される記憶装置を含む。記憶装置はその記憶装置中の可変位置に配置される複数のサイズ可変バッファをストアする。サイズ可変バッファの各々は、発生器によって出力されたグラフィクスコマンドを消費器へ分配するためにそれを受けかつ一時的にストアする。

【００１３】この発明によって提供される別の局面によれば、消費器は複数のバッファの少なくともアクティブな１つへ書き込むことはできないが、発生器とは独立して複数のバッファのそのアクティブな少なくともの１つのためのライトポインタを維持する。発生器は複数のバッファの第１バッファに関連する発生器リードポインタおよび発生器ライトポインタを与え、消費器は、その同じバッファに関連する消費器リードポインタおよび消費器ライトポインタを独立して維持する。消費器は、アクティブなバッファから消費器が読み出すことに応じて消費器リードポインタをインクリメントし、インクリメントした消費器リードポインタが消費器ライトポインタに対して所定の関係を有するときそのアクティブなバッファからの読出を中断する(suspend) 。消費器は発生器がアクティブなバッファへ書き込むことに応答して消費器ライトポインタを選択的にインクリメントする。

【００１４】この発明の他の局面によれば、バッファは、発生器が記憶装置中のどこかへストアしたグラフィクスコマンドのセットを消費器が消費し、どこかにストアされたグラフィクスコマンドを消費した後にバッファからのグラフィクスコマンドの消費を再開するように消費器を制御するリードコマンドを含む。このリードコマンドは、表示リストの開始アドレスおよび長さを特定する。リードコマンドは、その特定した開始アドレスで開始する特定した長さの表示リストを読み出すように消費器を制御する。

【００１５】この発明の他の局面によれば、複数のバッファの任意のものは循環的なもしくは直線的な先入れ先出し（ＦＩＦＯ）アクセスを提供する。

【００１６】この発明の他の局面によれば、複数のバッファの任意のものは発生器および消費器の両方へ同時に選択的に付属され、もしくはバッファの１つは発生器に付属され他のバッファが消費器に付属される。

【００１７】この発明によって提供されるなおも他の局面によれば、発生器は複数のバッファの各々のサイズを割り付ける。そのような割付は、グラフィクスコマンドの少なくとも１フレームを各バッファがストアすることができるようになされる。

【００１８】この発明の他の局面によれば、発生器は複数のバッファのいずれかへ中断点を書き込むことができる。消費器はこの中断点に遭遇したことに応じてグラフィクスコマンドの消費を中断することができる。

【００１９】この発明のさらに他の局面によれば、各バ

ッファは発生器がバッファの位置を上書きしたことを示すオーバフローステータス表示器(indicator)を提供する。

【００２０】この発明のなおも他の局面によれば、ステータスレジスタもしくは他の表示器が複数のバッファの少なくとも１つのステータスを表示することができる。ステータスレジスタは、たとえば、

・発生器ライトポインタの位置、

・発生器リードポインタの位置、

・消費器ライトポインタの位置、および

・消費器リードポインタの位置

を表示することができる。

【００２１】この発明のさらに他の局面によれば、グラフィクスシステムは

・グラフィクスコマンドを受けかつ一時的にストアする記憶バッファ、

・バッファ中へグラフィクスコマンドを書き込み、そのバッファに関連して発生器ライトポインタおよび発生器リードポインタを維持する発生器、および

・バッファ中にストアされたグラフィクスコマンドを消費し、発生器ライトポインタとは独立している消費器ライトポインタおよび発生器リードポインタとは独立している消費器リードポインタを維持する消費器

を含む。

【００２２】この発明のなおも他の局面によれば、グラフィクスシステムは発生器ライトポインタに基づいてバッファ中へグラフィクスコマンドを書き込むグラフィクスコマンド発生器、および消費器リードポインタに基づいてバッファからグラフィクスコマンドを読み出すグラフィクスコマンド消費器を含む。発明のこの局面によれば、消費器ライトポインタは消費器によって独立して維持され、発生器がバッファ中へ書き込んだ有効データの範囲を表示する。消費器は、消費器リードポインタが消費器ライトポインタに対して所定の関係を有することに応じてバッファからグラフィクスコマンドを消費するのを中止する。

【００２３】この発明によって提供されるなおも他の局面によれば、インタラクティブなグラフィクスシステムは、アプリケーションを実行するプロセサモジュール、グラフィクスプロセサモジュール、およびプロセサモジュールならびにグラフィクスプロセサモジュールに結合される少なくとも１つのメモリを含む。プロセサモジュールとグラフィクスプロセサモジュールとの間のグラフィクスコマンドのフローを制御する方法は

・ＦＩＦＯバッファの各々のサイズを特定するアプリケーションの制御の下で、メモリ中に可変数のＦＩＦＯバッファを動的に確立するステップ、

・複数のＦＩＦＯバッファ中の少なくとも第１バッファへグラフィクスコマンドを書き込むようにアプリケーションがプロセサモジュールを制御するステップ、および

・第１ＦＩＦＯバッファからグラフィクスコマンドを読み出すようにグラフィクスプロセサモジュールを制御するグラフィクスコマンドをアプリケーションがグラフィクスプロセサモジュールに送るステップ

を含む。

【００２４】プロセサモジュールは複数のバッファの第１バッファに関連してプロセサモジュールリードポインタおよびプロセサモジュールライトポインタを与える。グラフィクスプロセサモジュールは、第１バッファに関連してグラフィクスプロセサモジュールリードポインタおよびグラフィクスプロセサモジュールライトポインタを独立して維持する。グラフィクスプロセサモジュールは、グラフィクスプロセサモジュールが第１バッファから読み出すたび毎にグラフィクスプロセサモジュールリードポインタをインクリメントし、そのグラフィクスプロセサモジュールリードポインタがグラフィクスプロセサモジュールライトポインタと所定の関係を有するときその第１バッファからの読出を中断する。グラフィクスプロセサモジュールはプロセサモジュールが第１バッファへ書き込むことに応答してグラフィクスプロセサモジュールライトポインタを選択的に自動的にインクリメントする。

【００２５】この発明のなおも他の局面によれば、グラフィクスデータのフローを制御する方法は

・各々が複数の記憶位置を有する複数のサイズ可変ＦＩＦＯバッファ中へグラフィクスデータを書き込むステップ、

・複数の記憶位置の少なくとも１つに関連して中断点を設定するステップ、

・複数のバッファから所定の順序でグラフィクスデータを読み出すステップ、

・中断点に関連する少なくとも１つの記憶位置に遭遇したことに応答して読出ステップを一時的に中断し、割込を発生するステップ、および

・割込クリアコマンドの受信に応答して読出ステップを再開するステップ

を含む。

【００２６】この発明によって提供されるなおもその他の局面によれば、グラフィクスシステムは

・グラフィクスコマンドを受けかつ一時的にストアする記憶装置、

・記憶装置中のどこかへストアされているグラフィクスコマンドの第１セットおよびグラフィクスコマンドの第２セットを参照するリードコマンドを含むコマンドを記憶装置中のバッファへ書き込む発生器、および

・バッファ中へストアされたグラフィクスコマンドの第１セットを消費し、リードコマンドへの遭遇に応答して、グラフィクスコマンドの第２セットを消費し、そしてバッファからの付加的なコマンドを引き続いて消費する消費器

を含む。
【００２７】この発明の上述の目的，その他の目的，特徴および利点は、図面を参照して行う以下の実施例の詳細な説明から一層明らかとなろう。
【００２８】
【実施例】図１は対話型（インタラクティブ）３Ｄコンピュータグラフィクスシステム５０の一例を示す。システム５０は対話型３Ｄビデオゲームをステレオ音声とともにプレイするのに用いられ得る。これはまた多様な他のアプリケーションにも用いられ得る。
【００２９】この実施例において、システム５０は３次元世界のディジタル表現ないしモデルをインタラクティブにかつリアルタイムに処理することができる。システム５０は、任意の視点から、その世界の一部または全部を表示することができる。たとえば、システム５０は、手持ちコントローラ５２ａおよび５２ｂまたは他の入力デバイスからのリアルタイム入力に応答して、視点をインタラクティブに変化できる。このことによって、ゲームプレーヤは、その世界内もしくは外の誰かの目を通してその世界を見ることができる。システム５０は、リアルタイム３Ｄインタラクティブ表示を必要としないアプリケーション（たとえば２Ｄ表示の発生やおよび／またはノンインタラクティブ表示）に使用できるが、高品質の３Ｄ映像を非常に速く表示する能力は、非常にリアルでエキサイティングなゲームプレイや他のグラフィクスインタラクションを創造するのに使用され得る。
【００３０】システム５０を用いてビデオゲームまたは他のアプリケーションをプレイするために、ユーザはまず、主ユニット５４を、カラーテレビ５６または他の表示装置に、両者の間にケーブル５８を接続することによって、接続する。主ユニット５４はカラーテレビ５６を制御するためのビデオ信号およびオーディオ信号を発生する。ビデオ信号はテレビジョン画面５９上に表示されている映像を制御するものであり、オーディオ信号はテレビのステレオスピーカ６１Ｌおよび６１Ｒを通して音声として再生される。
【００３１】ユーザはまた主ユニット５４を電源につなぐ必要がある。この電源は従来のＡＣアダプタ（図示せず）であってよく、そのＡＣアダプタは家庭用の標準的な壁ソケットに差し込まれ、家庭用電源を、主ユニット５４を駆動するのに適した低いＤＣ電圧信号に変換する。他の実施例ではバッテリが用いられてもよい。
【００３２】ユーザは主ユニット５４を制御するために手持ちコントローラ５２ａおよび５２ｂを用いる。コントロール６０は、たとえば、３Ｄ世界内においてテレビ５６に表示されているキャラクタが移動すべき方向（上または下、左または右、近づいてまたは遠ざかって）を指示するために使用され得る。コントロール６０は、また他のアプリケーションのための入力（たとえばメニュー選択，ポインタ／カーソル制御，その他）を与える。コントローラ５２は多様な形態をとり得る。この実施例においては、図示されるコントローラ５２は、各々ジョイスティック，押しボタンおよび／または方向スイッチのようなコントロール６０を含む。コントローラ５２は、ケーブルによって、もしくは電磁波（たとえば電波または赤外線）を介してワイヤレスで、主ユニット５４に接続され得る。
【００３３】ゲームのようなアプリケーションをプレイするために、ユーザはビデオゲームもしくはプレイしたいと思う他のアプリケーションをストアしている適宜の記憶媒体６２を選択し、その記憶媒体を主ユニット５４のスロット６４に差し込む。記憶媒体６２は、たとえば、特別にエンコードされおよび／または記号化された光学的ならびに／もしくは磁気的ディスクであってよい。ユーザは主ユニット５４をオンするために電源スイッチ６６を操作し、主ユニットがその記憶媒体６２にストアされているソフトウェアに基づいてビデオゲームもしくは他のアプリケーションを実行し始めるようにする。ユーザは主ユニットに入力を与えるためにコントローラ５２を操作する。たとえば、コントロール６０を操作することによってゲームもしくは他のアプリケーションをスタートさせる。他のコントロール６０を動かすことによって、動画キャラクタを異なる方向に移動させ、または３Ｄ世界におけるユーザの視点を変化させる。記憶媒体６２にストアされている具体的なソフトウェアによって、コントローラ５２上の種々のコントロール６０は異なる時間で異なる機能を達成することができる。
全体システムの例
図２はシステム５０の例示的なコンポーネントのブロック図であり、重要なコンポーネントは、
・主プロセサ（ＣＰＵ）１１０，
・主メモリ１１２，および
・グラフィクス／オーディオプロセサ１１４
を含む。
【００３４】この実施例において、主プロセサ１１０（たとえばＩＢＭパワーＰＣ７５０の改良版）は、手持ちコントローラ５２（および／または他の入力デバイス）からの入力をグラフィクス／オーディオプロセサ１１４を通して受ける。主プロセサ１１０はユーザ入力にインタラクティブに応答し、光ディスクドライブのような大容量記憶媒体アクセス装置１０６を介して、たとえば外部記憶媒体６２から供給されるビデオゲームもしくは他のプログラムを実行する。一例として、ビデオゲームプレイの状況では、主プロセサ１１０は、多様なインタラクティブ制御機能に加えて、衝突検出および動画処理を実行する。
【００３５】この実施例では、主プロセサ１１０は３Ｄグラフィクス／オーディオコマンドを発生し、それらをグラフィクス／オーディオプロセサ１１４に送る。グラフィクス／オーディオプロセサ１１４はこれらのコマン

ドを処理し、ディスプレイ５９上での可視映像を生成し、ステレオスピーカ６１Ｒおよび６１Ｌもしくは他の適宜の音声発生デバイス上でのステレオ音声を生成する。

【００３６】実施例のシステム５０はビデオエンコーダ１２０を含み、このビデオエンコーダは、グラフィクス／オーディオプロセサ１１４からの映像信号を受けて、その映像信号をコンピュータモニタや家庭用テレビ５６のような標準的な表示装置上での表示に適したアナログおよび／またはディジタルビデオ信号に変換する。システム１００はまたオーディオコーデック（圧縮器／伸長器）１２２を含み、このオーディオコーデックはディジタル化されたオーディオ信号を圧縮しかつ伸長するとともに、必要に応じてディジタルオーディオ信号のフォーマットとアナログオーディオ信号のフォーマットとの間で変換を行う。オーディオコーデック１２２はバッファ１２４を介してオーディオ入力を受けることができ、処理（たとえば、プロセサが生成したおよび／または大容量記憶媒体アクセス装置１０６のストリームオーディオ出力を介して受信した他のオーディオ信号とのミキシング）するために、そのオーディオ入力をグラフィクス／オーディオプロセサ１１４に与える。この実施例におけるグラフィクス／オーディオプロセサ１１４は、オーディオタスクに利用可能なオーディオメモリ１２６にオーディオ関連情報をストアすることができる。グラフィクス／オーディオプロセサ１１４は、結果的に得られるオーディオ出力信号を、圧縮およびアナログ信号への変換のために、オーディオコーデック１２２に与え、したがってそのオーディオ出力信号が（たとえばバッファアンプ１２８Ｌおよび１２８Ｒを介して）スピーカ６１Ｌおよび６１Ｒによって再生され得る。

【００３７】グラフィクス／オーディオプロセサ１１４はシステム１００内に存在するであろう種々の付加的なデバイスと通信する能力を有する。たとえば、パラレルディジタルバス１３０は大容量記憶媒体アクセス装置１０６および／または他のコンポーネントと通信するために用いられる。シリアル周辺バス１３２は多様な周辺機器または、たとえば、

・ＰＲＯＭおよび／またはＲＴＣ（リアルタイムクロック）１３４，

・モデム１３６もしくは他のネットワークインタフェース（それはシステム１００を、プログラム命令および／またはデータがダウンロードもしくはアップロードされ得るインターネットあるいは他のディジタルネットワークのようなテレコミュニケーションネットワーク１３８に接続する），および

・フラッシュメモリ１４０

を含む他のデバイスと通信する。別の外部シリアルバス１４２は、付加的な拡張メモリ１４４（たとえばメモリカード）もしくは他のデバイスと通信するために使用され得る。コネクタが種々のデバイスをバス１３０，１３２および１４２に接続するために使用され得る。

グラフィクス／オーディオプロセサの例

図３は実施例のグラフィクス／オーディオプロセサ１１４を示すブロック図である。或る実施例においては、グラフィクス／オーディオプロセサ１１４はシングルチップＡＳＩＣであってよい。この実施例においては、グラフィクス／オーディオプロセサ１１４は、

・プロセサインタフェース１５０，

・メモリインタフェース／コントローラ１５２，

・３Ｄグラフィクスプロセサ１５４，

・オーディオディジタル信号プロセサ（ＤＳＰ）１５６，

・オーディオメモリインタフェース１５８，

・オーディオインタフェース／ミキサ１６０，

・周辺コントローラ１６２，および

・表示コントローラ１６４

を含む。

【００３８】３Ｄグラフィクスプロセサ１５４はグラフィクス処理タスクを実行する。オーディオディジタル信号プロセサ１５６はオーディオ処理タスクを実行する。表示コントローラ１６４は主メモリ１１２からの映像情報にアクセスし、表示装置１０２上での表示のためにその映像情報をビデオエンコーダ１２０に与える。オーディオインタフェース／ミキサ１６０はオーディオコーデック１２２をインタフェースし、また異なるソースからのオーディオ（たとえば、大容量記憶媒体アクセス装置１０６からのオーディオストリーム，オーディオＤＳＰ１５６の出力，およびオーディオコーデック１２２を通して受ける外部オーディオ入力）をミックスすることができる。プロセサインタフェース１５０は主プロセサ１１０およびグラフィクス／オーディオプロセサ１１４の間のデータおよび制御インタフェースを提供する。

【００３９】メモリインタフェース１５２はグラフィクス／オーディオプロセサ１１４とメモリ１１２との間のデータおよび制御インタフェースを提供する。この実施例においては、主プロセサ１１０は、プロセサインタフェース１５０およびグラフィクス／オーディオプロセサ１１４の一部であるメモリインタフェース１５２を介して、主メモリ１１２にアクセスする。周辺コントローラ１６２はグラフィクス／オーディオプロセサ１１４と上で述べた種々の周辺機器との間のデータおよび制御インタフェースを提供する。オーディオメモリインタフェース１５８はオーディオメモリ１２６とのインタフェースを提供する。

グラフィクスパイプラインの例

図４は図３の３Ｄグラフィクスプロセサ１５４をより詳細に示すグラフィクス処理システムを示す。この３Ｄグラフィクスプロセサ１５４は、とりわけ、コマンドプロセサ２００および３Ｄグラフィクスパイプライン１８０

を含む。主プロセサ１１０はデータストリーム（たとえばグラフィクスコマンドストリームおよび表示リスト）をコマンドプロセサ２００に通信する。主プロセサ１１０はメモリレイテンシを最小化するために２レベルキャッシュ１１２を有し、さらにまたグラフィクス／オーディオプロセサ１１４に向けられたキャッシュされていないデータストリームのための書込収集(write-gathering)バッファ１１１を有する。この書込収集バッファ１１１は部分キャッシュラインを全キャッシュラインに集め、バスの最大使用時に、グラフィクス／オーディオプロセサ１１４からのデータを１つのキャッシュラインに送る。

【００４０】コマンドプロセサ２００は主プロセサ１１０からの表示コマンドを受け、それらを解剖し、メモリコントローラ１５２を介して共用メモリ１１２からのそのコマンドを処理するに必要な付加的なデータを入手する。コマンドプロセサ２００は、２Ｄおよび／または３Ｄ処理およびレンダリングのために、頂点コマンドのストリームをグラフィクスパイプライン１８０に与える。グラフィクスパイプライン１８０はこれらのコマンドに基づいて映像を生成する。結果として得られた映像情報は、表示コントローラ／ビデオインタフェースユニット１６４によるアクセスのために主メモリ１２０に転送され得て、この映像情報は表示装置１５６上にパイプライン１８０のフレームバッファ出力を表示する。

【００４１】図5はグラフィクスプロセサ１５４を用いて実行される処理を図解的に示すブロックロジックフロー図である。主プロセサ１０は、グラフィクスコマンドストリーム２１０，表示リスト２１２および頂点アレイ２１４を主メモリ１１２にストアし、ポインタをバスインタフェース１５０を介してコマンドプロセサ２００に送る。主プロセサ１１０は主メモリ１１０内に割り付けられた１つ以上のグラフィクスＦＩＦＯバッファ２１０にグラフィクスコマンドをストアする。このコマンドプロセサ２００は、

・同期／フロー制御および負荷バランスのためにグラフィクスコマンドを受けかつバッファするオンチップＦＩＦＯメモリバッファ２１６を介して主メモリ１１２からのコマンドストリーム，

・オンチップコールＦＩＦＯメモリバッファ２１８を介して主メモリ１１２からの表示リスト２１２，および

・コマンドストリームからおよび／または主メモリ１１２の頂点アレイ２１４からの頂点アトリビュートを頂点キャッシュ２２０を介して

取り込む。

【００４２】コマンドプロセサ２００はコマンド処理動作２００ａを実行し、そのコマンド処理動作２００ａはアトリビュート形式を浮動小数点フォーマットに変換し、結果的に得られた完全頂点ポリゴンデータをレンダリング／ラスタライゼーションのためにグラフィクスパイプライン１８０に与える。プログラマブルメモリ調停回路１３０（グラフィクスメモリ要求調停回路：図４）は、グラフィクスパイプライン１８０，コマンドプロセサ２００および表示コントローラ／ビデオインタフェースユニット１６４の間での共用主メモリ１１２へのアクセスを調停する。

【００４３】図4は、グラフィクスパイプライン１８０が

・変換ユニット３００，

・セットアップ／ラスタライザ４００，

・テクスチャユニット５００，

・テクスチャ環境ユニット６００，および

・ピクセルエンジン７００

を含むことを示す。

【００４４】変換ユニット３００は多様な２Ｄおよび３Ｄ変換および他の動作３００ａ（図５）を実行する。変換ユニット３００は変換処理３００ａに用いられるマトリクスをストアするための１つ以上のマトリクスメモリ３００ｂを含む。変換ユニット３００は、入来する頂点毎のジオメトリをオブジェクト空間からスクリーン空間へ変換し、そして入来するテクスチャ座標を変換しかつ投影テクスチャ座標（３００ｃ）を計算する。変換ユニット３００はまたポリゴンクリッピング／カリング(clipping/culling)３００ｄを実行する。変換ユニット３００ｂによってまた達成されるライティング(lighting)処理３００ｅが、この実施例では８つまでの独立した照明(light：ライト)について、頂点毎にライティング計算を行う。変換ユニット３００は、エンボス(embossed)タイプのバンプマッピング効果およびポリゴンクリッピング／カリング動作（３００ｄ）のために、テクスチャ座標を発生する（３００ｃ）。

【００４５】セットアップ／ラスタライザ４００はセットアップユニットを含み、このセットアップユニットは、変換ユニット３００からの頂点データを受け、三角形セットアップ情報を、エッジラスタライゼーション，テクスチャ座標ラスタライゼーションおよびカラーラスタライゼーションを実行する１つ以上のラスタライザユニット（４００ｂ）に送る。

【００４６】テクスチャユニット５００は、オンチップテクスチャメモリ（ＴＭＥＭ）５０２を含んでもよく、たとえば、

・主メモリ１１２からのテクスチャ５０４の検索、

・たとえばマルチテクスチャ処理，ポストキャッシュテクスチャ伸長，テクスチャフィルタリング，エンボシング，投影テクスチャの使用を通しての陰影付け，およびアルファトランスパーレンシおよびデプスを用いるＢＬＩＴを含むテクスチャ処理（５００ａ）、

・バンプマッピング，偽(pseudo)テクスチャおよびテクスチャタイル(tiling)効果（５００ｂ）のためのテクスチャ座標変位を計算するバンプマップ処理、および

・間接テクスチャ処理（５００ｃ）
を含むテクスチャリングに関連する種々のタスクを実行する。
【００４７】テクスチャユニット５００はテクスチャ環境処理（６００ａ）のためにフィルタされたテクスチャ値をテクスチャ環境ユニット６００に出力する。テクスチャ環境ユニット６００は、ポリゴンおよびテクスチャカラー／アルファ／デプスをブレンドし、また逆レンジベース(reverse range based)のフォグ効果を達成するために、テクスチャフォグ処理（６００ｂ）を実行する。テクスチャ環境ユニット６００はたとえばカラー／アルファ変調，エンボシング，詳細テクスチャ，テクスチャスワッピング，クランピングおよびデプスブレンディングに基づく多様な他の環境関連機能を実行する多数のステージを提供する。
【００４８】ピクセルエンジン７００はデプス（ｚ）比較（７００ａ）およびピクセルブレンディング（７００ｂ）を実行する。この実施例では、ピクセルエンジン７００はデータを埋め込み（オンチップ）フレームバッファメモリ７０２にストアする。グラフィクスパイプライン１８０は、フレームバッファおよび／またはテクスチャ情報をローカルにストアするために１つ以上の埋め込みＤＲＡＭメモリ７０２を含む。ｚ比較７００ａは、現在有効なレンダリングモードに依存して、グラフィクスパイプライン１８０におけるより早い段階で実行される（たとえば、ｚ比較は、もしアルファブレンディングが要求されていないならば早くに実行され得る）。このピクセルエンジン７００は表示コントローラ／ビデオインタフェースユニット１６４による主メモリ１１２へのアクセスのために、オンチップフレームバッファ７０２を周期的に書き込むコピー動作７００ｃを含む。このコピー動作７００ｃはまた動的テクスチャ合成効果のために、埋め込みフレームバッファ７０２の内容を主メモリ１１２中のテクスチャにコピーするために使用され得る。アンチエイリアシング(anti-aliasing：エイリアス補正)および他のフィルタリングがコピー動作中に実行され得る。最終的に主メモリ１１２にストアされるグラフィクスパイプライン１８０のフレームバッファ出力は、表示コントローラ／ビデオインタフェースユニット１６４によってフレーム毎に読み出される。表示コントローラ／ビデオインタフェース１６４は表示装置５６上での表示のためにディジタルＲＧＢピクセル値を与える。
共用メモリ中に割り付けられたＦＩＦＯバッファ
この実施例において、グラフィクス／オーディオプロセサ１１４のオンボードのコマンドＦＩＦＯバッファ２１６（これは小規模のデュアルポートＲＡＭのストリーミングバッファであってよい）は、それ自身によって、プロセサ１１０とグラフィクスパイプライン１８０との間の負荷バランスをとる良好なジョブを行うには小さすぎる。このことによって、グラフィクス／オーディオプロセサ１１４が大きなプリミティブをレンダリングしているときには、プロセサ１１０は停止されることになる。この問題を改善するために、プロセサ１１０とグラフィクス／オーディオプロセサ１１４との間で共用される主メモリ１１２の一部をコマンドＦＩＦＯバッファ２１０として用いる。バッファ２１０を用いることによって、主プロセサ１１０およびグラフィクスプロセサ１１４がほぼそれらのピークレート(peak rate) で並列的に動作することができる。
【００４９】並列処理を達成するためにバッファ２１０を用いる（少なくとも）２つの方法があり、中間モード(immediate mode)およびマルチバッファモード(multi-buffer mode) である。１つのバッファ２１０が主プロセサ１１０およびグラフィクスプロセサ１１４の両方へ付属させられるとき、システム５０は中間モードで動作している。主プロセサ１１０がグラフィクスコマンドをバッファ２１０へ書き込むと、グラフィクスプロセサ１１４はそれらを順に処理する。書込が読出を追い越すのを防ぎかつ循環的なバッファ動作を提供するためにバッファ２１０のリードおよびライトポインタを第１アドレスへラップさせるためにハードウェアサポートがフロー制御ロジックを与える。
【００５０】好ましい実施例においては、また、マルチバッファモードにおいて、グラフィクス／オーディオプロセサ１１４が異なるバッファ２１０（１）から読出している間１つのバッファ２１０を主プロセサ１１０へ接続することができる。この場合、バッファ２１０（１）および２１０（２）は、いずれの特定のバッファ２１０へも同時的な読出および書込がないので、従来のＦＩＦＯというよりもバッファのように管理される。バッファの動的なメモリ管理が望まれるなら、マルチバッファモードが使用され得る。
【００５１】図６は共用メモリ１１２の一部が発生器１１０および消費器１１４の間でのグラフィクス（およびオーディオ）コマンドをバッファリングするためにどのようにして割り当てられて多数のＦＩＦＯコマンドバッファ２１０（１），２１０（２），…，２１０（ｎ）を提供するかを示す。図６に示す例において、バッファ２１０の各々は主プロセサ１１０からのグラフィクス（および／またはオーディオ）コマンドを受け、それらのコマンドをグラフィクス／オーディオプロセサ１１４へ与える。主プロセサ１１０はこれらのバッファ２１０としての使用のために、主メモリ１１２の一部を割り当てる。主メモリの領域を記述するバッファデータ構造子(structure) が、主プロセサ１１０上で実行中のアプリケーションによって割り付けられる。
【００５２】主プロセサ１１０はライトポインタ８０２を用いてバッファ中へグラフィクスコマンドを書き込む。グラフィクス／オーディオプロセサ１１４はリード

ポインタ８０４を用いてバッファ２１０からコマンドを読み出す。ライトポインタ８０２およびリードポインタ８０４は同じのもしくは異なるバッファをポイント（指示）することができる。この方法において、同じバッファ２１０が主プロセサ１１０およびグラフィクス／オーディオプロセサ１１４の両方へ同時に「付属」させられ、もしくは異なるバッファが発生器および消費器に異なる時間で付属させられ得る。

【００５３】図６に示すマルチバッファの例において、主プロセサ１１０およびグラフィクス／オーディオプロセサ１１４は、「その」ＦＩＦＯバッファ２１０が割り当てられている場所で必ずしも一致する必要はない。図示する例において、主プロセサ１１０はグラフィクスコマンドを書き込むための現在のバッファとしてバッファ２１０（２）を用い、これに対して、グラフィクス／オーディオプロセサ１１４はグラフィクスコマンドを入手するために現在のバッファとして異なるバッファ２１０（１）を使用する。バッファ２１０は主プロセサ１１０，グラフィクス／オーディオプロセサ１１４あるいはその両方へ動的に付属させられ得る。バッファが主プロセサへ付属させられるとき、主プロセサはグラフィクスコマンドをバッファ２１０へ書き込む。実施例において、任意の一時に主プロセサ１１０へ付属させられる１つでかつ１つだけのバッファ２１０がある。バッファがグラフィクスプロセサ１１４へ付属されるとき、グラフィクスプロセサはその付属されたバッファ２１０からグラフィクスコマンドを読み出しかつ処理する。この実施例においては、ただ１つのバッファ２１０だけが任意の一時にグラフィクスプロセサ１１４へ付属され得る。

独立した消費器および発生器のリードおよびライトポインタ

グラフィクスコマンド発生器として作用する主プロセサ１１０は、それが付属されているバッファ２１０（２）から読み出す必要はないが、図６の例に示すように発生器リードポインタ８０６を維持する。同じように、グラフィクス／オーディオプロセサ１１４はグラフィクスコマンドの消費器として動作し、それゆえに、それが付属されているバッファ２１０（１）へ書き込む必要はないけれども、図６の例においては消費器ライトポインタ８０８を維持する。これらの付加的なポインタ８０６および８０８によって発生器および消費器がそれに付属されている各バッファ２１０を独立的に維持することができる。

【００５４】主プロセサ１１０によって維持されている付加的なポインタ８０６およびグラフィクス／オーディオプロセサ１１４によって維持されている付加的なポインタ８０８はオーバラップ検出を与えるために使用される。これらの余分なポインタはバッファ２１０内において有効データがどこに存在するかを示す。たとえば、主プロセサ１１０はそれに付属されているバッファ２１０（２）を循環バッファとして取り扱い、ライトポインタがバッファ８１２の「終り」に達すると、バッファ８１０の「開始」にそのライトポインタを「ラップ(wrap)」する。しかしながら、消費器ライトポインタ８０２が消費器リードポインタ８０６へ遭遇すると、消費器は、グラフィクス／オーディオプロセサ１１４がまだ読み出していない有効な、先に書き込まれたデータを上書きするのを回避するために、消費器はその付属するバッファ２１０（２）への書込を中止する。同じように、グラフィクス／オーディオプロセサ１１４は、それの付属するバッファ２１０（１）からグラフィクス命令を漸進的に読み出すとき、それのリードポインタ８０４を継続的にインクリメントするが、リードポインタ８０４がライトポインタ８０８に遭遇すると、消費器はライトポインタをバッファ２１０（１）内の最終有効データを示すように用いているので、このインクリメントの手続きを中止する。

【００５５】ポインタ８０２，８０４，８０６および８０８はバッファ２１０内の任意の位置をポイントすることができる。有効データは、したがってこれらのバッファ内の任意の場所に存在し、必ずしもバッファの始めまたは終りである必要はない。実際に、もしバッファ２１０が循環モード(circular mode) で動作すると、バッファの終りは始めにラップされ、そしてそれゆえにバッファは論理的には連続ループであるので、「開始」または「終了」の概念はない。

【００５６】図７は独立した消費器および発生器のリードおよびライトポインタの簡単化した説明を提供する。図７の例において、消費器１１４はそれに付属されているバッファ２１０（１）からグラフィクスコマンドを読み出すために自動インクリメントリードポインタ８０４を用いる。消費器１１４はまた、バッファ２１０（１）中の最終有効グラフィクスコマンドをポイントする消費器ライトポイント８０８を維持する。この例において、消費器１１４はバッファ２１０（１）からグラフィクスコマンドを読み出し続け、各グラフィクスコマンドを読み出した後に、ライトポインタが示す同じ位置（図８参照）をポイントするまで、それのリードポインタ８０４をインクリメントする。リードポインタ８０４がライトポインタ８０８が示す位置に隣接する位置を示すように消費器１１４がそのリードポインタ８０４をインクリメントしたとき、消費器は、バッファ２１０（１）から有効グラフィクスコマンドのすべてを読み出しそしてそれゆえにバッファがエンプティになったことを「知る」。この状態は、消費器１１４が発生器１１０（もしバッファ２１０（１）が同時に発生器に付属されているなら）からのさらなるグラフィクスコマンドを待つ必要があるか、もしくは（もしマルチバッファが有効であるなら）、消費器が読出を開始するべく異なるバッファ２１０に向ける必要があることを示している。

【００５７】同じように、発生器はそれに付属されているバッファ２１０（２）中へグラフィクスコマンドを書き続け、同じように、発生器リードポインタ８０６が示す位置の直前であるバッファの位置をライトポインタがポイントするまで、発生器ライトポインタ８０２を自動的にインクリメントする（図９参照）。この例において、ライトポインタ８０２とリードポインタ８０６との間の一致（実際にはごく接近して）は、バッファ２１０（２）がフルであることを示す。マルチバッファが有効であれば、発生器はこの時点でバッファ２１０（２）への書込を中止し、それを「セーブ」（閉じる）し、消費器に対してその「閉じた」バッファの内容を（今または後に）読み出すように指令し、主メモリ１１２中に割り当てることができるなおも他のバッファ２１０へ付加的なグラフィクスコマンドを書き込むのを開始する。発生器１１０および消費器１１４が同じバッファへ付属されると、そのバッファへさらなるコマンドを書き込む前に、発生器は、消費器がコマンドを読み出すまで待つ必要がある。以下に説明するように、このような頻繁な状況の切換を回避するために、好ましい実施例はプログラマブルなヒステリシス効果を提供でき、それは発生器がバッファへの書込の再開を許容される前に或る量によってバッファがエンプティにされることを要し、消費器がバッファからの読出の再開を許容される前に或る量によってバッファがフルになる必要がある。

【００５８】好ましい実施例においては、主プロセサ１１０は、３２バイト転送でそれに付属されるバッファ２１０へグラフィクスコマンドを書き込む。主プロセサ１１０は書込収集バッファ／機能１１１（図４参照）を提供し、それは、グラフィクスコマンドを自動的に３２バイトワードへパックする(pack)。グラフィクスプロセサ１１４は３２バイト転送においてそれが付属されるバッファ２１０からグラフィクスコマンドを読み出す。

ＦＩＦＯバッファからの表示リストのコール

図１０は好ましい実施例によって提供される技術の一例であり、それによって、関数コールとほとんど同じように、ＦＩＦＯバッファ２１０の入力が表示リストをコールすることができる。この例において、コマンド８９０はメモリ中のどこかにストアされた表示リスト２１２をコールするグラフィクスコマンドＦＩＦＯ２１０中に挿入される。このコマンド８９０に遭遇すると、グラフィクスプロセサ１１４はＦＩＦＯバッファ２１０からのグラフィクスコマンドの読出を一時的中止し、代わって、主メモリ１１２中のどこかにストアされている表示リスト２１２からのコマンドを読み出し始める。表示リスト２１２の終りに達すると、グラフィクスプロセサ１１４はグラフィクスＦＩＦＯ２１０からの次の順次的なコマンドを読み出すためにリターンする。この技術は、主プロセサ１１２が各フレーム毎に表示リストを再書込する必要がなく、多数のフレームが同じ表示リスト２１２をコールできるという点で非常に有用である（たとえば、フレームからフレームへ静止したままであるジオメトリをレンダリングするような場合）。

【００５９】図１１ないし図１３は、グラフィクスコマンドＦＩＦＯ２１０へ書き込むことによって主プロセサ１１０が自動的に表示リスト２１２を生成することができることを示す。図１１に示すように、主プロセサ１１０は主メモリ１１２中に割り付けたグラフィクスコマンドＦＩＦＯ２１０へグラフィクスコマンドストリームを書き込むことによって開始する。この書込プロセスにおける任意の時点で、主プロセサ１１０はＦＩＦＯバッファ２１０中へ「表示リスト開始(Begin DisplayList)」コマンド８９０を挿入することができ、そのコマンドによって、主プロセサからの別の書込を表示リスト２１２へ向けることができる。図１３は、主プロセサ１１０が表示リスト２１２の書込を終了すると、「表示リスト終了(End Display List)」コマンドを発生することを示し、このコマンドは表示リストを自動的に終了させ、主プロセサのコマンドストリーム出力をＦＩＦＯバッファ２１０へ自動的に向け直す効果を有する。主プロセサ１１０が向け直し可能な「消火ホース」コールストリーム出力を与え、その出力がＦＩＦＯバッファ２１０、表示リスト２１０および同じもしくは異なるＦＩＦＯバッファ２１２へグラフィクスコマンドを流出させることができることを思い浮かべることができる。この方法によって生成された表示リスト２１２はメモリ１１２中に留まり、いくつかのフレームあるいはフレーム部分に亘って静止したままであるイメージの一部のために再使用され得る。

具体例の詳細

グラフィクス／オーディオプロセサ１１４のグラフィクスインタフェースユニット部分２０２へのプロセサであるコマンドプロセサ２００は、主メモリ１１２中のＦＩＦＯバッファ２１０を管理するための制御ロジックを含む。図１４はその具体例を示す。図示する例において、ＣＰＵ１１０がグラフィクス／オーディオプロセサ１１４へ書き込んだすべてのものが主メモリ１１２へ送られる。グラフィクス／オーディオプロセサ１１４へ付属するグラフィクスＦＩＦＯ２１０へ割り付けられた主メモリ１１２の一部を規定する２つのレジスタがあり、それはＦＩＦＯベース(BASE)レジスタ８２２、およびＦＩＦＯ先頭(TOP) レジスタ８２４である。

【００６０】ＦＩＦＯ＿ＢＡＳＥレジスタ８２２はＦＩＦＯ２１０のベースアドレスを規定する。ＦＩＦＯ＿ＴＯＰレジスタ８２４はＦＩＦＯにおける最終(last)アドレスを規定する。

【００６１】コマンドプロセサ２００はハードウェア中においてＦＩＦＯ２１０のためのリードおよびライトポインタのトラッキングを保持する。ＦＩＦＯ中に書き込まれたすべてのデータがキャッシュライン(cache line)

のサイズにされるので、有効バイトのトラッキングを保持する必要はない。ライトポインタ８０８はＦＩＦＯ＿ＢＡＳＥとＦＩＦＯ＿ＴＯＰとの間にあるアドレス中にキャッシュラインが書き込まれる毎に３２バイトによってインクリメントされる（最上位５ビットは０）。ＦＩＦＯ２１０の読出が一度に１キャッシュラインで行われる。リードポインタはキャッシュラインが読み出された後、３２によってインクリメントされる。

【００６２】最初に、リードポインタ８０４およびライトポインタ８０８が同じ位置を示すように初期化され、それはＦＩＦＯがエンプティであることを意味する（図８参照）。ＦＩＦＯフル状態は、（リードポインター１）＝（ライトポインタ）である（図９参照）。ライトポインタ８０８はそれがＦＩＦＯ＿ＴＯＰへ達した後にＦＩＦＯ＿ＢＡＳＥ２０４（２）へラップされる。リードポインタ８０４もまたそれがＦＩＦＯ＿ＴＯＰ８２４へ達するときラップされる。リードポインタ８０４はハードウェアによって制御され、ラップされた場合においてもライトポインタ８０８を追い越さないようにされている。プロセサ１１０上で動いているアプリケーションによって、ライトポインタ８０８は、ラップされた後にも、リードポインタ８０４より大きくならないようにされる。

【００６３】２つ（またはそれ以上）の異なるフレームからのデータは同じＦＩＦＯ２１０中へ存在することができる。第１フレームが内蔵ＤＲＡＭ７０２へコピーアウトされる前にコマンドプロセサ２００が第２フレームを実行するのを防止するために中断点機構が用いられ得る。ＦＩＦＯ中断点（レジスタ）８３２がイネーブルされると、コマンドプロセサ２００はＣＰ＿ＦＩＦＯ＿ＢＲＫレジスタを読み出さない。ＣＰＵ１１０はフレームの終わりでこのレジスタ８３２をプログラムすることができる。ＣＰＵ１１０はグラフィクス／オーディオプロセサ１１４上の書込バッファをフラッシュしなければならず、次いで、ＦＩＦＯライトポインタ８０８を読み出す。そして、ＣＰＵ１１０はＦＩＦＯ中断点レジスタ８３２へ値を書き込み、中断点をイネーブルする。

【００６４】ＦＩＦＯ２１０のサイズが１フレーム中に送られたすべてのデータを保持するに十分大きければ、図９に示すＦＩＦＯフル状態は決して発生しない。しかしながら、このことは主メモリ１１２の２Ｍないし４Ｍバイトをフ ＩＦＯバッファ２１０のために割り付けることを意味する。アプリケーションの開発者はＦＩＦＯ２１０のために大きなメモリを使用したいとは思わないであろう。その場合、アプリケーションはフロー制御技術を実現する。レジスタ８２６および８２８はそのようなフロー制御を提供するために使用され得る。フロー制御は、主メモリ１１２中のキャッシュラインの数がＦＩＦＯ＿ＨＩＣＮＴ８２６より大きくなったときグラフィクス／オーディオプロセサ１１４がＣＰＵ１１０へ割込を発生させることによって、この実施例中において行われる。プロセサ１１０は割込を取り入れ、ＦＩＦＯ中のキャッシュラインの数がＦＩＦＯ＿ＬＯＣＮＴ８２８より小さくなるまで、スピン(spin)するかもしくは他の非グラフィクスタスクを行う。そのようなヒステリシス効果を提供する理由は、割込のオーバヘッドが高く、ＦＩＦＯ２１０の内容が「高水位マーク（high water mark：ハイウォータマーク）」以下になったことをただチェックすることによって割込ルーチンに入るか割込ルーチンを出るかを行うことを望まないからである。割込はＦＩＦＯのカウントがＬＯＣＮＴ８２８以下になったとき発生され得る。この方法によって、アプリケーションは割込がかけられたとき他のタスクを行いリターンする。

ＦＩＦＯバッファの割付の例

好ましい実施例において、グラフィクスＡＰＩはスタティックなFXFifoObj の構造子を内部的に宣言する。この構造子はGXInitがコールされたとき初期化される。

【００６５】

【表１】

GXFifoObj* GXInit (void* base, u32 size);

【００６６】ＦＩＦＯベースポインタは、この好ましい実施例においては３２ビットに合わされている。アプリケーションはＦＩＦＯのためにメモリを割り付ける責任がある。割付のためのサイズのパラメータはバイトにおけるＦＩＦＯのサイズである（最小ＦＩＦＯサイズは６４Ｋバイトであり、サイズは３２バイトの倍数である）。デフォルトによって、GXInitが中間モードグラフィクスのＦＩＦＯをセットアップすると、すなわち、ＣＰＵ１１０およびグラフィクスプロセサ１１４がＦＩＦＯに付属されると、リードおよびライトポインタはベースポインタに初期化され、高水位および低水位マーク（low water mark：ローウォータマーク）がイネーブルされる。GXInitは初期化したGXFifoObj へのポインタをアプリケーションへリターンする。

【００６７】アプリケーションがマルチバッファモードで動作することを望むなら、次いで、追加のＦＩＦＯが割り付けられなければならない。任意の数のそのような追加のＦＩＦＯバッファ２１０が割り付けられ得る。アプリケーションは各追加のＦＩＦＯ毎にメモリに割り付け、GXFifoObj を同じように初期化する。次に例示する関数はGXFifoObj を初期化するために使用され得る。

【００６８】

【表２】

```
void GXInitFifoBase(
                GXFifoObj*      fifo,
                void*           base,
                u32             size);
void GXInitFifoPtrs(
                GXFifoObj*      fifo
                void*           read_ptr,
                void*           write_ptr );
void GXInitFifoLimits(
                GXFifoObj*      fifo,
                u32             hi_water_mark,
                u32             lo_water_mark );
```

【００６９】通常、アプリケーションはＦＩＦＯリードおよびライトポインタをＦＩＦＯのベースアドレスへ初期化する必要があるだけである。一旦初期化されると、システムハードウェアはリードおよびライトポインタを自動的に制御する。

ＦＩＦＯの付属およびセーブ

ＦＩＦＯが一旦初期化されると、それはＣＰＵ１１０またはグラフィクスプロセサ１１４もしくはその両方へ付属される。ただ１つのＦＩＦＯが同時にＣＰＵ１１０およびグラフィクスプロセサ１１４のいずれかに付属され得る。ＦＩＦＯがＣＰＵ１１０に付属されると、ＣＰＵはＦＩＦＯに対してGXコマンドを発行する。ＦＩＦＯがグラフィクスプロセサ１１４に付属されたとき、プロセサ１１４はＦＩＦＯからグラフィクスコマンドを読み出すためにイネーブルされる。次に例示する関数がＦＩＦＯを付属させる。

【００７０】

【表３】

```
void GXSetCPUFifo( GXFifoObj* fifo );
void GXSetGPFifo( GXFifoObj* fifo );
GXFifoObj* GXGetCPUFifo ( void ) ;
GXFifoObj* GXGetGPFifo ( void );
```

【００７１】どのＦＩＦＯオブジェクトが現在これらの例示的な関数によって付属されたかを問い合わせる。

【００７２】

【表４】

```
GXFifoObj* GXGetCPUFifo ( void );
GXFifoObj* GXGetGPFifo ( void );
```

【００７３】マルチバッファモードにおいてＣＰＵ１１０がGXコマンドの書込を終了すると、ＦＩＦＯは新たなＦＩＦＯに切り換える前に「セーブ」されなければならない。次に例示する関数がＣＰＵのＦＩＦＯを「セーブ」する。

【００７４】

【表５】

```
void GXSaveCPUFifo ( FXFifoObj* fifo );
```

【００７５】ＦＩＦＯがセーブされると、ＣＰＵの書込収集バッファ１１１がフラッシュされてすべてのグラフィクスコマンドが主メモリ１１２へ書き込まれる。さらに、現在のＦＩＦＯのリードおよびライトポインタがGXFifoObj の構造子中にストアされる。

【００７６】グラフィクスプロセサ１１４のためにはセーブ関数はないということに留意されたい。グラフィクスプロセサが付属されると、グラフィクスコマンドは

・ＦＩＦＯがエンプティになる、

・ＦＩＦＯ中断点に遭遇する、あるいは

・ＧＰが先に獲得される

まで読み出され続ける。

ＦＩＦＯステータス

次の例示の関数はＦＩＦＯおよびＧＰ（グラフィクスパイプライン）のステータスを読み出すために使用され得る。

【００７７】

【表６】

```
                GXBool*         overhi,
                GXBool*         underlo,
                u32*            fifo_cnt,
                GXBool*         cpu_write,
                GXBool*         gp_read,
                GXBool*         fifowrap );

void GXGetGPStatus(
                GXBool*         overhi,
                GXBool*         underlow,
                GXBool*         readIdle,
                GXBool*         cmdIdle,
                GXBool*         brkpt );
```

【００７８】GXGetFifoStatus は特定のＦＩＦＯのステータスを獲得する。もし、ＦＩＦＯが現在ＣＰＵ１１０に付属されていると、パラメータcpu_writeはGX_TRUEである。ＦＩＦＯが現在グラフィクスプロセサ１１４に付属されていると、パラメータgp_readはGX_TRUEである。ＦＩＦＯがＣＰＵ１１０もしくはグラフィクスプロセサ１１４のいずれかに付属されているとき、ステータスはハードウェアの状態から直接に読み取られる。ＦＩＦＯが付属されていなければ、ステータスはGXFifoObj から読み出される。GXGetFifoStatus は特定されたＦＩＦＯがオーバフローしたかあるいはそこへの書込に十分な部屋(room)を有するかを報告する。一般的に、ハードウェアはＦＩＦＯがオーバフローしたこと、すなわちデータの量がＦＩＦＯのサイズを超えたことを検出することはできない。

【００７９】ＦＩＦＯのオーバフローを検出する一般的な方法はないけれども、ハードウェアは、ＣＰＵライトポインタがＦＩＦＯの先頭に達したことを検出すること

ができる。この状態が生じると、「fifowrap」の引数(argument)がGX_TRUEにリターンする。この「fifowrap」の引数は、もしＣＰＵのライトポインタが常にＦＩＦＯのベースに初期化されるとすると、ＦＩＦＯのオーバフローを検出するために使用され得る。「fifowrap」は、ＦＩＦＯが現在ＣＰＵ１１０に付属されていれば、セットされる。

【００８０】GXGetGPStatus はグラフィクスプロセサ１１４のステータスを（それに付属されているＦＩＦＯに拘わらず）を取り込むために使用され得る。新たなグラフィクスプロセサのＦＩＦＯを付属させる前に必要な最小限の条件はグラフィクスプロセサ１１４がアイドル状態(idle)になるのを待つことである（しかし付加的な制約が存在する）。アンダーフローおよびオーバフローのステータスはライトポインタがハイウォータマークおよびローウォータマークに関連する場所を示す。

ＦＩＦＯのフロー制御

ＦＩＦＯがＣＰＵおよびグラフィクスプロセサの両方に付属されているとき（中間モード）、ＦＩＦＯがフルになり過ぎたときにはＣＰＵがコマンドの書込を停止するように注意しなければならない。「ハイウォータマーク」はグラフィクスコマンドがもはやＦＩＦＯへ書き込めなくなるであろう前に如何にＦＩＦＯのフルを取り込むかを規定する。好ましい実施例においては、ＣＰＵ内に１６Ｋバイトまでのグラフィクスコマンドがバッファリングされ得て、そのためにハイウォータマークを（ＦＩＦＯのサイズ－１６Ｋバイト）に設定することが望ましい。

【００８１】ハイウォータマークに遭遇すると、プログラムは中断されるが、オーディオのような他の割込駆動タスクは作動する。プログラマはマルチスレッドプログラムにおいてどの具体的なスレッドが中断されるべきであるかを特定することを望む。

【００８２】「ローウォータマーク」はプログラム（またはスレッド）が継続できる前に「ハイウォータマーク」に達した後にＦＩＦＯのエンプティを取り込まなければならない。ローウォータマークは（ＦＩＦＯのサイズ／２）に設定されることが望ましい。新たなコマンドデータの量がハイウォータマークに接近して留まっているときにプログラムが何らかのレジスタをポーリングしたり定常的にオーバフロー割込を受け付ける必要がないので、このローウォータマークはプログラムにおける頻繁な状況の切換を防止する。

【００８３】マルチバッファモードにおいては、ハイおよびローウォータマークはディスエーブルされる。ＦＩＦＯがＣＰＵ１１０へ付属されていてＣＰＵがＦＩＦＯが保持できるより以上のコマンドを書き込むときに、ライトポインタは最終アドレスからベースアドレスへラップされ戻される。ＦＩＦＯにある先のグラフィクスコマンドは上書きされる。ライトポインタがＦＩＦＯの先頭へラップするときを検出することはできる（それは、コマンドが送られる前にＦＩＦＯのライトポインタがＦＩＦＯのベースに初期化されている場合に限ってオーバフローを示す）。上で述べたGXGetFifoStatus を参照のこと。

【００８４】マルチバッファモードにおいてＦＩＦＯ（バッファ）のオーバフローを防止するために、ソフトウェアベースでチェックする方法が使用され得る。主プロセサ１１０上で実行中のプログラムはバッファサイズのそれ自身のカウンタを保持しなければならず、そして任意のグループのコマンドがバッファへ付加される前に、プログラムは部屋があるかどうかをチェックして判断する。もし部屋が利用可能であれば、グループのサイズがバッファサイズへ加えられ得る。もし部屋が利用可能でなければ、バッファはフラッシュされて新たなものが割り付けられる。

表示リストコールの使用

好ましい実施例においてＦＩＦＯバッファ２１０から表示リストをコールするために、アプリケーションはまず、表示リストをストアするためのメモリ中のスペースを割り付ける。メモリエリアが一旦セットアップされると、アプリケーションは、たとえば次のものをコールする。

【００８５】

【表７】

```
void GXBeginDisplayList (
        void            *list
        u32             size);
```

【００８６】「リスト」の引数が表示リストがストアされる位置の開始アドレスであり、「サイズ」の引数が表示リストコマンドを書き込むための割り付けられたスペースにおいて利用可能なバイト数を示す場合、システムがオーバフローをチェックすることができる。

【００８７】「GXBeginDisplayList」が一旦コールされると、別のGXコマンドが通常のコマンドＦＩＦＯに代わって表示リストへ書き込まれる。「GXEndDisplayList」コマンドは表示リストの終了へ信号を送り、先に向けられていたＦＩＦＯへコマンドストリームをリターンさせる。「GXEndDisplayList」コマンドはまた、生成された表示リストの実際のサイズを、好ましい実施例においては３２バイトの倍数としてリターンする。

【００８８】実施例において、表示リストは入れ子(nest)されることはない。このことは、GXBeginDisplayListが一旦発行されると、GXEndDisplayListコマンドがやって来るまでに他のGXBeginDisplayListまたはGXCallDisplayList コマンドを発行することは違法であることを意味する。しかしながら、別の実施例においては、任意の所望の入れ子レベルに表示リストを入れ子することはできる。

グラフィクスＦＩＦＯ関数の例　【００８９】
以下に例示する関数はグラフィクスＦＩＦＯの管理を提供する。　【表８】

```
GXSetFifoBase:
引数
(Argument)
        u32         BasePtr;        //主メモリに FIFO のベースアドレスをセット
        u32         Size;           //FIFO のバイトサイズ(３２バイトマルチプル)
        GXBool      Set Defaults    //FIFO の状態のデフォルトをセット
```

【００９０】グラフィクスＦＩＦＯの制限を設定する。この関数は初期化時にコールされる。ＦＩＦＯアドレスはグラフィクスパイプラインがフラッシュされない限り変更されることはない。SetDefaultのフラグがセットされると、ＦＩＦＯはリセットされ（すなわち、リード／ライトポインタをＦＩＦＯのベースへ）、そして割込がディスエーブルされる。デフォルトによって、ハイウォータマークがサイズの２／３に設定され、ローウォータマークがサイズの１／３に設定される。
【００９１】
【表９】

```
GXSetFifoLimits:
Argument
        u32         HiWaterMark;    //FIFO のためのハイウォータ(Hi-water)マーク
        u32         LoWaterMark;    //ローウォータ(Low water)マーク
        u32         RdBreakMark;    //リードポインタの中断点(break point)
```

【００９２】この関数はＦＩＦＯの制限をセットする。リードポインタがローウォータマーク以下になったときもしくはライトポインタがハイウォータマーク以上になったとき、グラフィクスハードウェアはＣＰＵへ割込をかける。RdBreakMark がリードポインタ中断点を設定するために使用される。
【００９３】
【表１０】

```
GXSetInterrupts:
Argument
        GXBool      Underflow;      //ローウォータマーク割込みの能動化／不能動化
        GXBool      Overflow;       //ハイウォータマーク割込みの能動化／不能動化
        GXBool      BreakPoint;     //FIFO リード中断点の能動化／不能動化
```

【００９４】ＦＩＦＯに関連する割込をイネーブルしもしくはディスエーブルする。BreakPointは、先のフレームがまだコピーされているときにＣＰによってＦＩＦＯの読出をフォールトするために使用され得る。
【００９５】
【表１１】

```
GXClearInterrupts:
Argument:
        GXBool      Underflow;      //ローウォータマーク割込みのクリア
        GXBool      Overflow;       //ハイウォータマーク割込みのクリア
        GXBool      BreakPoint      // FIFO リード中断点のクリア
```

【００９６】保留中の割込をクリアする。　【表１２】
【００９７】

```
GXSetFifoPtrs:
Argument:
        u32         WritePtr;       //FIFO のためにライトポインタをセット
        u32         ReadPtr;        //リードポインタをセット
```

【００９８】ＦＩＦＯリードおよびライトポインタをセットする。これらのポインタはハードウェアによって維持される。この関数はハードウェア値を無視する（たとえば、表示リストのコンパイルのために）。
【００９９】
【表１３】

GXGetFifoStatus:
Argument:

```
GXBool    *UnderFlow;    //FIFOのカウントがローウォータマーク以下である
GXBool    *OverFlow;     //FIFOのカウントがハイウォータマーク以上である
GXBool    *BreakPoint;   //FIFOのリードポインタが中断点である
u32       *FifoCount;    //FIFOのキャッシュライン数(３２バイト)
```

【０１００】ＦＩＦＯのステータスおよびカウントをリターンする。

表示リスト関数の例

表示リストは予めコンパイルされたコマンドのアレイであり、グラフィクスパイプラインのためのデータである。次に例示するコマンドは表示リストを操作するためにＦＩＦＯバッファ２１０中へ挿入される。

【０１０１】

【表１４】

GXBeginDisplayList:
Argument:

```
void*     BasePtr;   //表示リストデータをストアするためのバッファのアドレス
u32       nBytes;    //バッファのサイズ
```

【０１０２】この関数は表示リストを生成しかつ開始させる。ＡＰＩは表示リストモードにされる。これに後続するすべてのＡＰＩ関数は、表示リスト関数の任意のものを除いて、EndDisplayListまでコールし、グラフィクスパイプラインに代わって表示リストバッファへそれらのデータおよびコマンドを送る。表示リストはこの実施例においては入れ子されない。すなわち、BeginDisplayListおよびEndDisplayListの間ではどんな表示リスト関数もコールされることはない。表示リストのためのメモリはアプリケーションによって割り付けられる。

【０１０３】

【表１５】

GXEndDisplayList:
Argument:

None.

Return:

```
u32       nBytes     //表示リストのために使用するバイト数
```

【０１０４】この関数は現在開かれている表示オブジェクトを終了させ、システムを中間モードに戻す。

【０１０５】

【表１６】

GXCallDisplayList:
Argument:

```
void*     BasePtr;   //表示リストデータをストアするためのバッファのアドレス
u32       nBytes;    //バッファのサイズ
```

【０１０６】この関数は表示リストを実行する。

レジスタフォーマット

次のテーブルはＣＰＵ１１０によってアドレス可能なコマンドプロセサ２００における例示的なレジスタを示す。

【０１０７】

【表１７】

レジスタ名称(Name)	ビットフィールド：記述
CP_STATUS レジスタ 834	0: FIFO オーバフロー (fifo_count > FIFO_HICNT) 1: FIFO アンダフロー (fifo_count < FIFO_LOCNT) 2: FIFO リードユニットアイドル 3: CP アイドル 4: FIFO 中断点到達 (FIFO 中断点を不能動化することによってクリアされる)
CP_ENABLE レジスタ 836	0: FIFO リード能動化、リセット値 "0"が不能動化 1: FIFO 中断点能動化、リセット値 "0" が不能動化 2: FIFO オーバフロー割込み能動化、リセット値 "0" が不能動化 3: FIFO アンダフロー割込み能動化、リセット値 "0" が不能動化 4: FIFO ライトポインタインクリメント能動化、リセット値 "1" が不能動化 5: FIFO 中断点割込み能動化、リセット値 "0" が不能動化
CP_CLEAR レジスタ 838	0: クリア FIFO オーバフロー割込み 1: クリア FIFO アンダフロー割込み
CP_STM_LOW レジスタ 840	7:0 ３２バイトインクリメントにおけるストリーミングバッファロ･-ウォータマークのビット 7:0、デフォルト（リセット）値は "0x0000"
CP_FIFO_BASEL 822	15:5 メモリの FIFO ベ･-スアドレスのビット 15:5
CP_FIFO_BASE 822	9:0 メモリの FIFO ベースアドレスのビット 25:16
CP_FIFO_TOPL 824	15:5 メモリの FIFO トップアドレスのビット 15:5
CP_FIFO_TOPH 824	9:0 メモリの FIFO トップアドレスのビット 25:16
CP_FIFO_HICNTL 826	15:5 FIFO のハイウォータカウントのビット 15:5
CP_FIFO_HICNTH 826	9:0 FIFO のハイウォータカウントのビット 25:16
CP_FIFO_LOCNTL 828	15:5 FIFO のローウォータカウントのビット 15:5
CP_FIFO_LOCNTH 828	9:0 FIFO のローウォータカウントのビット 25:16
CP_FIFO_COUNTL 830	15:5 FIFO_COUNT (現在 FIFO へ入力する)のビット 15:5
CP_FIFO_COUNTH 830	9:0 FIFO_COUNT (現在 FIFO へ入力する)のビット 25:16
CP_FIFO_WPTRL 808	15:5 FIFO ライトポインタのビット 15:5
CP_FIFO_WPTRH 808	9:0 FIFO ライトポインタのビット 25:15
CP_FIFO_RPTRL 804	15:5 FIFO リードポインタのビット 15:5
CP_FIFO_RPTRH 804	9:0 FIFO リードポインタのビット 25:15
CP_FIFO_BRKL 832	15:5 FIFO リードアドレス中断点のビット 15:5
CP_FIFO_BRKH 832	9:0 FIFO リードアドレス中断点のビット 9:0

【０１０８】互換性のある他の実施例

上述のシステム５０のあるものは上で述べた家庭用ビデオゲームコンソールの構成以外としても実現できる。たとえば、或るものは、システム５０をエミュレートする異なる構成を有するプラットフォームもしくはそれと同等のものにおいて、システム５０のために書かれたグラフィクスアプリケーションや他のソフトウェアを実行させることができる。もし、他のプラットフォームがシステム５０のいくつかのもしくはすべてのハードウェアおよびソフトウェアリソースをエミュレートしシミュレートしおよび／または提供することができれば、その他のプラットフォームはそのソフトウェアを成功裏に実行することができる。

【０１０９】一例として、エミュレータがシステム５０のハードウェアおよび／またはソフトウェア構成（プラットフォーム）とは異なるハードウェアおよび／またはソフトウェア構成（プラットフォーム）を提供できる。そのエミュレータシステムは、それのためにアプリケーションソフトウェアが書かれているシステムのいくつかのもしくはすべてのハードウェアおよび／またはソフトウェアコンポーネンツをエミュレートしもしくはシミュレートするソフトウェアおよび／またはハードウェアコンポーネンツを含む。たとえば、エミュレータシステムはパソコンのような汎用ディジタルコンピュータを含み、それはシステム５０のハードウェアおよび／またはファームウェアをシミュレートするソフトウェアエミュレータプログラムを実行する。上述のオーディオシステムのＤＳＰ処理がパソコンによってエミュレートされ得る。

【０１１０】或る汎用ディジタルコンピュータ（たとえばＩＢＭやマッキントッシュのパソコンおよびそれらの同等物）は、ダイレクトＸ(DirectX)または他の標準的な３Ｄグラフィクスコマンドａｐｉｓに従った３Ｄグラフィクスパイプラインを提供する３Ｄグラフィクスカードを備える。それらはまた、音声コマンドの標準的なセットに基づいて高品質のステレオ音声を提供するステレオ音声カードを備える。エミュレータソフトウェアを実行するそのようなマルチメディアのハードウェアを備えるパソコンは、システム５０のグラフィクスおよび音声性能とほぼ等しい十分な性能を有する。エミュレータソフトウェアはパソコンプラットフォーム上のハードウェアリソースを制御して、それのためにゲームプログラマがゲームソフトウェアを書いた家庭用ビデオゲームコンソールプラットフォームの処理，３Ｄグラフィクス，音

声，周辺および他の能力をシミュレートする。

【０１１１】図１４はホストプラットフォーム１２０１，エミュレータコンポーネント１３０３および記憶媒体６２上のゲームソフトウェア実行可能バイナリ映像を用いる全体のエミュレーション処理を図解する。ホスト１２０１は、たとえばパソコン，ビデオゲームコンソールあるいは十分な計算力を有する任意の他のプラットフォームのような汎用または特定目的ディジタル計算装置である。エミュレータ１３０３はそのホストプラットフォーム１２０１上で走るソフトウェアおよび／またはハードウェアであり、記憶媒体６２からのコマンド，データおよび他の情報のそのホスト１２０１によって実行可能な形態へのリアルタイム変換を行う。たとえば、エミュレータ１３０３は記憶媒体６２からシステム５０によって実行されるように意図された「ソース」であるバイナリ映像プログラム命令を取り込み、これらのプログラム命令をホスト１２０１によって実行されもしくは処理され得るターゲットとなる形態に変換する。

【０１１２】一例として、ソフトウェアがＩＢＭパワーＰＣまたは他の特定のプロセサを用いるプラットフォーム上での実行のために書かれかつホスト１２０１が異なる（たとえばインテル）プロセサを用いるパソコンである場合、エミュレータ１２０３は記憶媒体１３０５からの１つのもしくは一連のバイナリ映像プログラム命令を取り込み、これらのプログラム命令を１つまたはそれ以上の同等のインテルのバイナリ映像プログラム命令に変換する。エミュレータ１２０３はまたグラフィクス／オーディオプロセサ１１４によって処理されるように意図されたグラフィクスコマンドおよびオーディオコマンドを取り込みかつ／あるいは生成し、そしてホスト１２０１上で利用可能なハードウェアおよび／またはソフトウェアグラフィクス／オーディオ処理リソースによって処理され得る形態にこれらのコマンドを変換する。一例として、エミュレータ１３０３はホスト１２０１の特別なグラフィクスおよび／または音声ハードウェア（たとえば標準的なダイレクトＸ，オープンＧＬおよび／または音声ＡＰＩｓ）によって処理され得るコマンドにこれらのコマンドを変換する。

【０１１３】システム５０の或るエミュレータは、上で述べたバッファリングおよびフロー制御技術のいくつかもしくはすべてを単に「無視」する。なぜなら、エミュレータは上で述べた実施例のハードウェア構成より非常に多いメモリリソースを持たなければならないからである。そのようなエミュレータは、典型的には、割付メモリリソースによるバッファ割付のための要求に応答するが、異なるフロー制御処理を提供するであろう。上で述べたステータスおよびフロー制御要求は、ハードウェアのエミュレートされた状態を維持しそのステータス要求に応答するためにその状態を使用することによってエミュレートされ得る。

【０１１４】上で述べたビデオゲームシステムの特徴のいくつかあるいはすべてを提供するために使用されるエミュレータ１３０３は、グラフィクスユーザインタフェース（ＧＵＩ）を備え、そのＧＵＩはエミュレータを使ったゲームの実行のための種々の動作やスクリーンモードの選択を単純化しもしくは自動化する。一例において、そのようなエミュレータ１３０３はさらに、ソフトウェアがそれのために元々意図されていたホストプラットホームに比べて増強された機能性を含む。

【０１１５】図１５はエミュレータ１３０３で用いるに適したエミュレーションホストシステム１２０１を図解的に示す。このシステム１２０１は処理ユニット１２０３およびシステムメモリ１２０５を含む。システムバス１２０７がシステムメモリ１２０５を含む種々のシステムコンポーネンツを処理ユニット１２０３に結合する。システムバス１２０７は多様なバスアーキテクチャのいずれかを用いるメモリバスもしくはメモリコントローラ，周辺バスおよびローカルバスを含むいくつかのタイプのバス構造の任意のものである。システムメモリ１２０７はＲＯＭ１２５２およびＲＡＭ１２５４を含む。起動中においてのようにパソコンシステム１２０１中のエレメント（要素）間に情報を伝送する手助けをする基本ルーチンを含む基本入力／出力システム（ＢＩＯＳ）１２５６がＲＯＭ１２５２中にストアされる。システム１２０１はさらに種々のドライブおよび関連のコンピュータ読出可能な媒体を含む。ハードディスクドライブ１２０９が（典型的には固定の）磁気ハードディスク１２１１から読み出しそれへ書き込む。付加的な（たぶんオプションとしての）磁気ディスクドライブ１２１３が着脱可能な「フロッピィ」または他の磁気ディスク１２５１から読み出しかつそれへ書き込む。光ディスクドライブ１２１７はＣＤ－ＲＯＭあるいは他の光学媒体のような着脱自在な光ディスク１２１９から読み出しかつそれへ書き込む。ハードディスクドライブ１２０９および光ディスクドライブ１２１７は、ハードディスクドライブインタフェース１２２１および光ディスクドライブインタフェース１２２５によって、システムバス１２０７にそれぞれ接続される。これらのドライブおよびその関連するコンピュータ読出可能な媒体は、パソコンシステム１２０１のためのコンピュータ読出可能な命令，データ構造，プログラムモジュール，ゲームプログラムおよび他のデータの不揮発性の記憶媒体を提供する。他の構成では、コンピュータによってアクセス可能なデータをストアすることができる他のタイプのコンピュータ読出可能な媒体（たとえば磁気カセット，フラッシュメモリカード，ディジタルビデオディスク，ベルヌーイカートリッジ，ＲＡＭ，ＲＯＭあるいはその他のもの）がまた使用できる。

【０１１６】エミュレータ１３０３を含む多数のプログラムモジュールがハードディスク１２１１，着脱可能な

磁気ディスク１２１５，光ディスク１２１９および／またはシステムメモリ１２０５のＲＯＭ１２５２および／またはＲＡＭ１２５４にストアされ得る。このようなプログラムモジュールはグラフィクス／音声ＡＰＩｓ，１つ以上のアプリケーションプログラム，他のプログラムモジュール，プログラムデータおよびゲームデータを提供するオペレーティングシステム（ＯＳ）を含む。ユーザは、キーボード１２２７，ポインティングデバイス１２２９，マイクロフォン，ジョイスティック，ゲームコントローラ，衛星アンテナ(satellite dish)，スキャナあるいはその他のもののような入力デバイスを通して、パソコンシステム１２０１にコマンドおよび情報を入力することができる。これらのそして他の入力デバイスは、システムバス１２０７に結合されたシリアルポートインタフェース１２３１を通して処理ユニット１２０３に接続され得るが、パラレルポート，ゲームポートファイヤワイヤバス(Fire Wire)もしくはユニバーサルシリアルバス（ＵＳＢ）のような他のインタフェースによって接続されてもよい。モニタまたは他のタイプの表示デバイスがまたビデオアダプタ１２３５のようなインタフェースを介してシステムバス１２０７に接続される。

【０１１７】システム１２０１はモデム１１５４またはインターネットのようなネットワーク１１５２を通しての通信を確立するための他のネットワークインタフェース手段を含む。内蔵もしくは外付けであってよいモデム１１５４はシリアルポートインタフェース１２３１を介してシステムバス１２３に接続される。システム１２０１がローカルエリアネットワーク１１５８を介して遠隔コンピュータ装置１１５０（たとえば他のシステム１２０１）と通信するのを許容するために、ネットワークインタフェース１１５６がまた設けられてもよい（もしくはそのような通信はダイヤルアップもしくは他の通信手段のようなワイドエリアネットワーク１１５２もしくは他の通信経路を介してもよい）。システム１２０１はプリンタのような周辺出力装置および他の標準的な周辺装置を含む。

【０１１８】一例では、ビデオアダプタ１２３５は、マイクロソフト(Microsoft)のダイレクトＸ７．０、または他のバージョンのような標準的な３Ｄグラフィクスアプリケーションプログラマインタフェースに基づいて発行された３Ｄグラフィクスコマンドに応答して、高速の３Ｄグラフィクスレンダリングを提供する３Ｄグラフィクスパイプラインチップセットを含んでもよい。１組のスピーカ１２３７はまた、バス１２０７によって与えられる音声コマンドに基づいて高品質ステレオ音声を生成するハードウェアおよび埋め込みソフトウェアを提供する従来の「音声カード」のような音声生成インタフェースを介して、システムバス１２０７に接続される。これらのハードウェア能力によって記憶媒体１３０５中にストアされているソフトウェアを再生するためにシステム１２０１に十分なグラフィクスおよび音声の速度性能を与えることができる。

【０１１９】上で参照したすべての書類をここで、参照によって取り入れる。

【０１２０】最も現実的かつ好ましい実施例であると現在考えられているものに関連してこの発明が説明されたが、この発明は開示された実施例に限定されるものではなく、逆に、特許請求の範囲内に含まれる種々の変形例や等価的な構成をカバーするように意図されていることを理解されたい。

【図面の簡単な説明】

【図１】図１はインタラクティブコンピュータグラフィクスシステムの実施例を示す全体図である。

【図２】図２は図１実施例のコンピュータグラフィクスシステムのブロック図である。

【図３】図３は図２に示す実施例のグラフィクス／オーディオプロセサのブロック図である。

【図４】図４は図３に示す実施例の３Ｄグラフィクスプロセサのブロック図である。

【図５】図４のグラフィクス／オーディオプロセサの例示的なロジックフロー図である。

【図６】図６はマルチバッファリングの例を示す。

【図７】図７は独立した消費器および発生器のリードならびにライトポインタの例を示す。

【図８】図８はエンプティおよびフルバッファ状態の例を示す。

【図９】図９はエンプティおよびフルバッファ状態の例を示す。

【図１０】図１０はＦＩＦＯバッファからの表示リストのコールの例を示す。

【図１１】図１１は表示リスト作成例を示す。

【図１２】図１２は表示リスト作成例を示す。

【図１３】図１３は表示リスト作成例を示す。

【図１４】図１４はＦＩＦＯ管理の実施例を示す。

【図１５】図１５は互換性のある実施例を示す。

【図１６】図１６は別の互換性のある実施例を示す。

【符号の説明】

５０　…インタラクティブ３Ｄコンピュータグラフィクスシステム

５４　…主ユニット

１１０　…主プロセサ

１１１　…書込収集バッファ

１１２　…主メモリ

１１４　…グラフィクス／オーディオプロセサ

１８０　…グラフィクスパイプライン

２００　…キャッシュ／コマンドプロセサ

２１０　…グラフィクスコマンドＦＩＦＯ

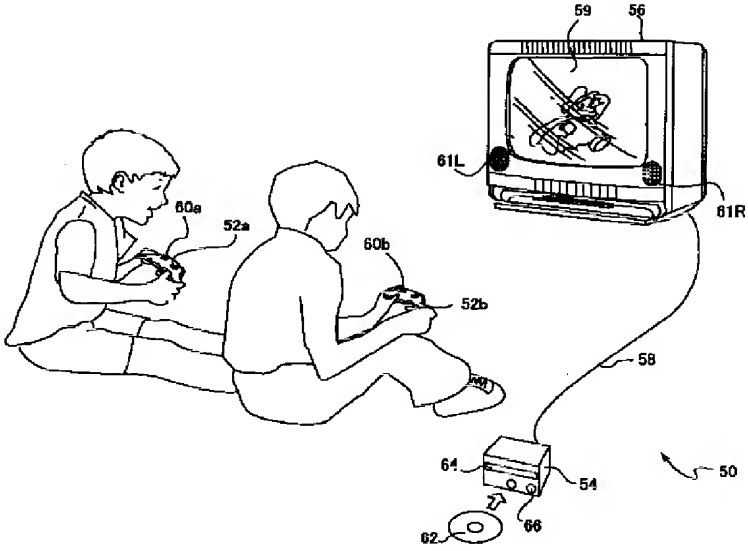
２１２　…表示リスト

８０２　…発生器ライトポインタ

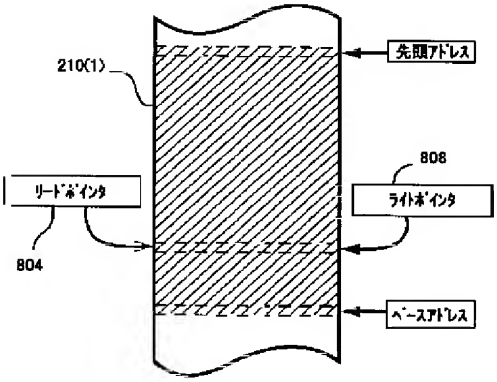
８０４　…発生器リードポインタ

８０６　…消費器ライトポインタ
８０８　…消費器リードポインタ
８９０　…中断点

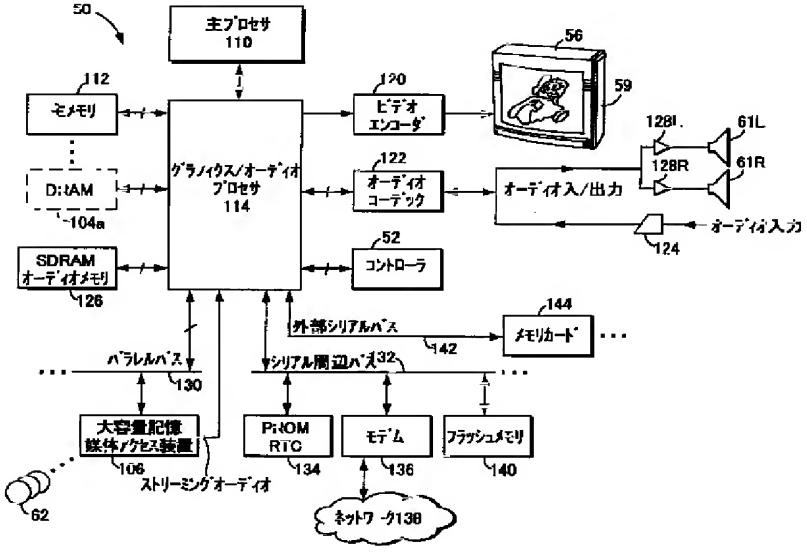
【図１】



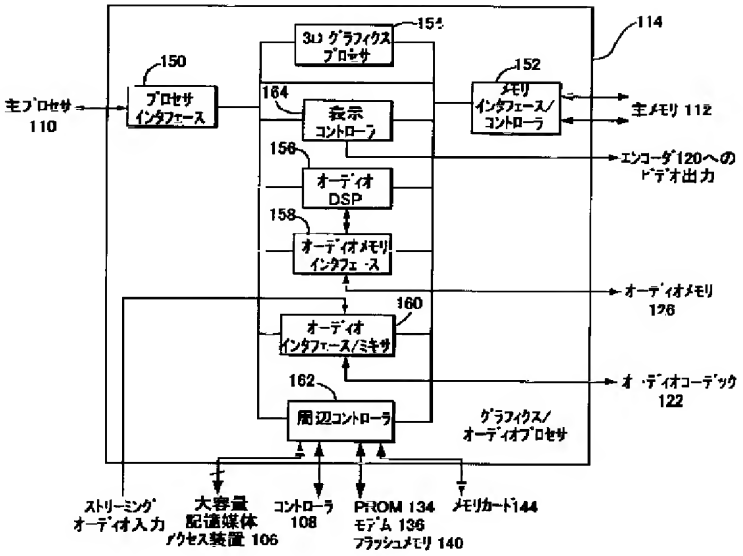
【図８】



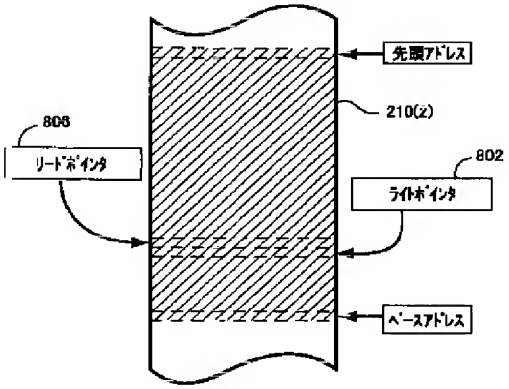
【図２】



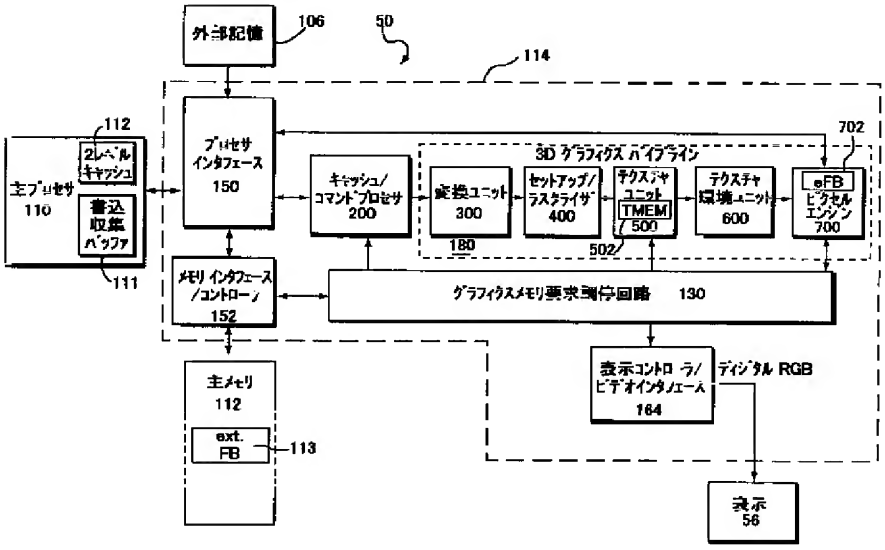
【図３】


主プロセサ
110
プロセサ
インタフェース
150
3D グラフィクス
プロセサ
154
表示
コントローラ
164
メモリ
インタフェース/
コントローラ
152
114
主メモリ 112
エンコーダ120への
ビデオ出力
オーディオ
DSP
156
オーディオメモリ
インタフェース
158
オーディオメモリ
176
オーディオ
インタフェース/ミキサ
160
オーディオコーデック
122
周辺コントローラ
162
グラフィクス/
オーディオプロセサ
ストリーミング
オーディオ入力
大容量
記憶媒体
アクセス装置 106
コントローラ
108
PROM 134
モデム 136
フラッシュメモリ 140
メモリカード144

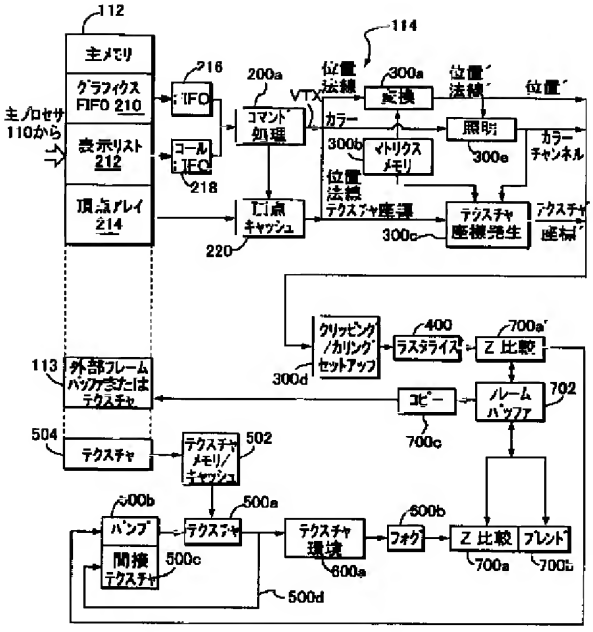
【図９】


先頭アドレス
210(z)
802
ライトポインタ
806
リードポインタ
ベースアドレス

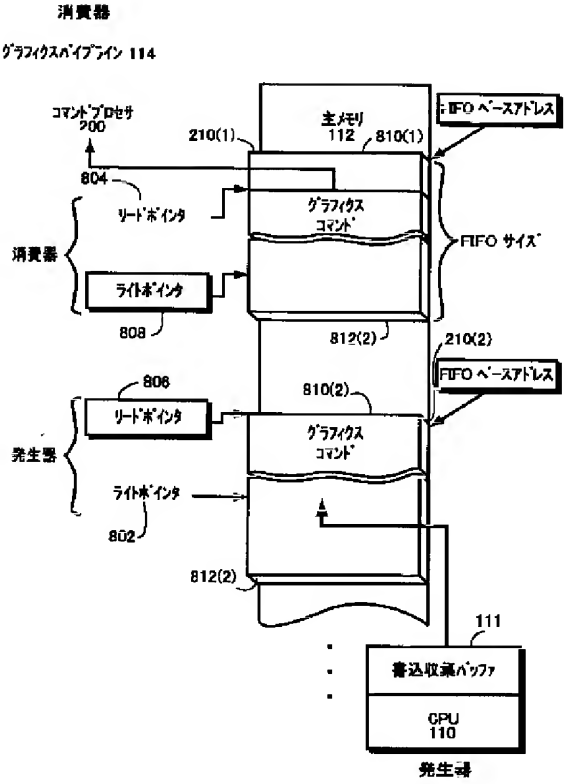
【図４】


外部記憶
106
50
114
112
2レベル
キャッシュ
主プロセサ
110
書込
収集
バッファ
111
プロセサ
インタフェース
150
キャッシュ/
コマンドプロセサ
200
3D グラフィクス パイプライン
変換ユニット
300
180
セットアップ/
ラスタライザ
400
テクスチャ
ユニット
TMEM
500
502
テクスチャ
環境ユニット
600
eFB
ピクセル
エンジン
700
702
メモリ インタフェース
/コントローラ
152
グラフィクスメモリ要求調停回路 130
主メモリ
112
ext.
FB
113
表示コントローラ/
ビデオインタフェース
164
ディジタル RGB
表示
56

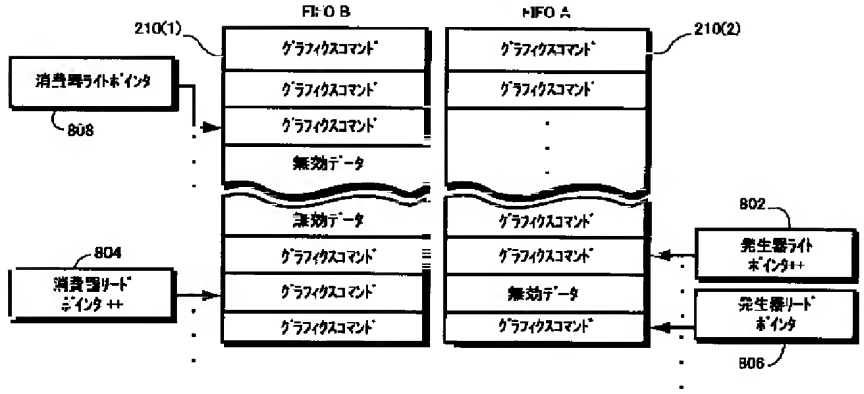
【図5】


主メモリ
グラフィクス FIFO 210
表示リスト 212
頂点アレイ 214
主プロセサ 110から
TFO
コール TFO
コマンド処理
VTX
位置 法線
変換
カラー
照明
マトリクス メモリ
カラーチャンネル
工点キャッシュ
テクスチャ座標
テクスチャ座標発生
クリッピング/カリング/セットアップ
ラスタライズ
Z 比較
外部フレームバッファまたはテクスチャ
コピー
ルームバッファ
テクスチャ
テクスチャメモリ/キャッシュ
バンプ
間接テクスチャ
テクスチャ環境
フォグ
Z 比較
ブレンド

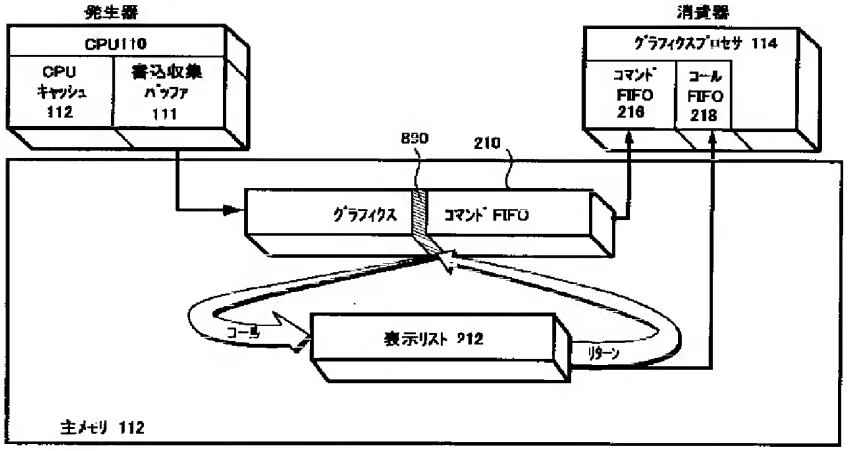
【図6】


消費器
グラフィクスパイプライン 114
コマンドプロセサ 200
主メモリ 112
FIFO ベースアドレス
リードポインタ
ライトポインタ
グラフィクスコマンド
FIFO サイズ
消費器
発生器
書込収集バッファ
CPU 110
発生器

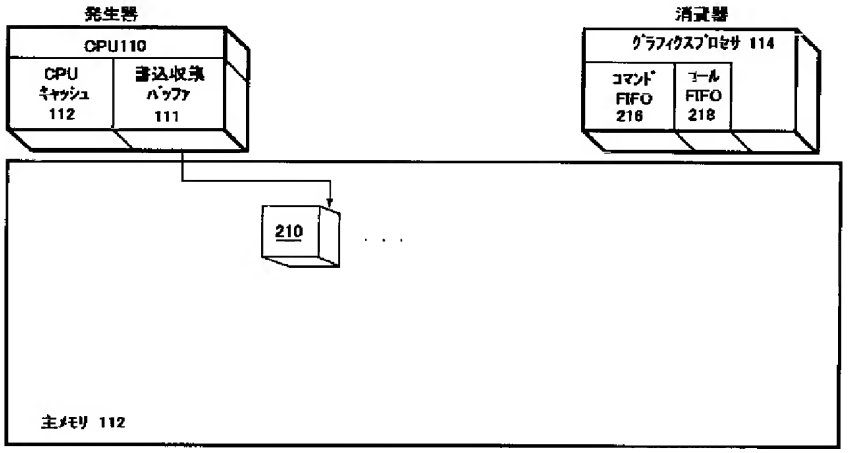
【図7】


FIFO B
FIFO A
消費器ライトポインタ
消費器リードポインタ ++
発生器ライトポインタ++
発生器リードポインタ
グラフィクスコマンド
無効データ

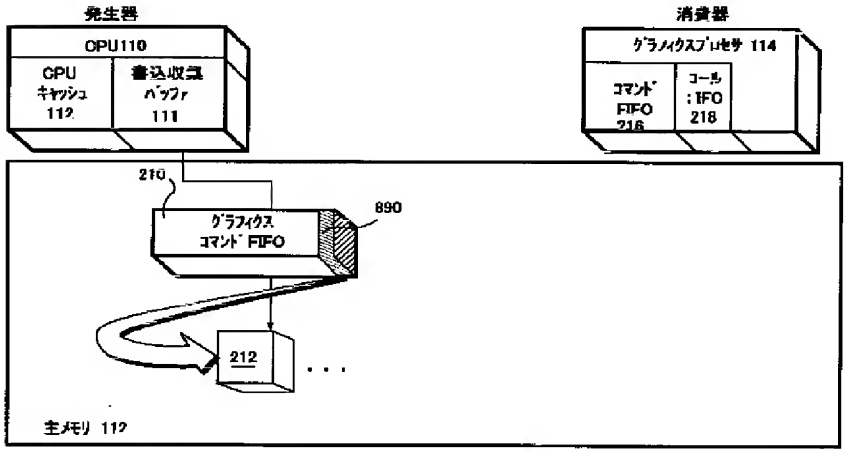
【図１０】


発生器
CPU110
CPU
ｷｬｯｼｭ
112
書込収集
ﾊﾞｯﾌｧ
111
消費器
ｸﾞﾗﾌｨｸｽﾌﾟﾛｾｻ 114
ｺﾏﾝﾄﾞ
FIFO
216
ｺｰﾙ
FIFO
218
890
210
ｸﾞﾗﾌｨｸｽ
ｺﾏﾝﾄﾞ FIFO
ｺｰﾙ
表示ﾘｽﾄ 212
ﾘﾀｰﾝ
主ﾒﾓﾘ 112

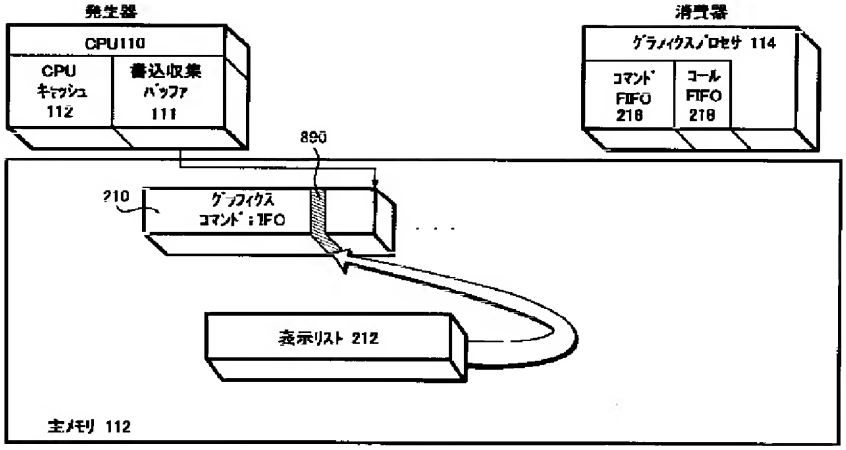
【図１１】


発生器
CPU110
CPU
ｷｬｯｼｭ
112
書込収集
ﾊﾞｯﾌｧ
111
消費器
ｸﾞﾗﾌｨｸｽﾌﾟﾛｾｻ 114
ｺﾏﾝﾄﾞ
FIFO
216
ｺｰﾙ
FIFO
218
210
主ﾒﾓﾘ 112

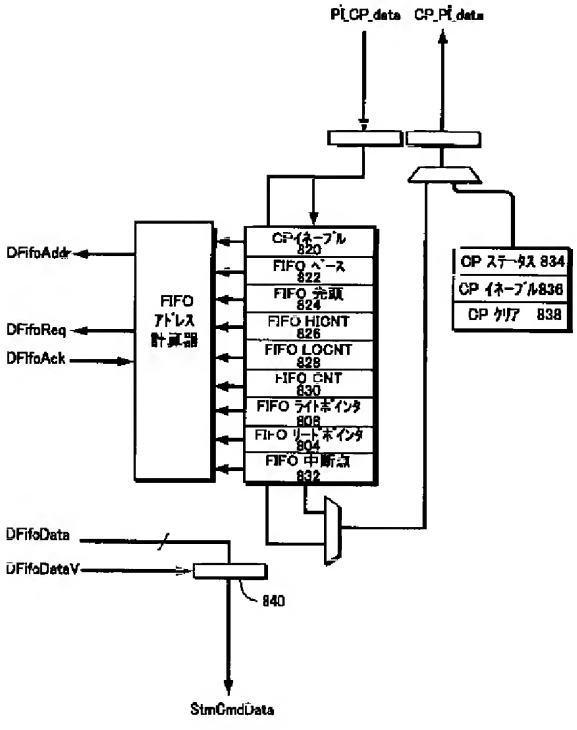
【図１２】


発生器
CPU110
CPU
ｷｬｯｼｭ
112
書込収集
ﾊﾞｯﾌｧ
111
消費器
ｸﾞﾗﾌｨｸｽﾌﾟﾛｾｻ 114
ｺﾏﾝﾄﾞ
FIFO
216
ｺｰﾙ
FIFO
218
210
890
ｸﾞﾗﾌｨｸｽ
ｺﾏﾝﾄﾞ FIFO
212
主ﾒﾓﾘ 112

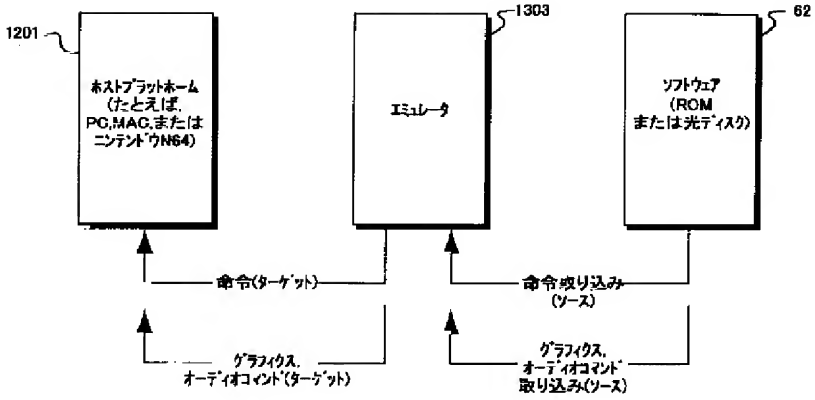
【図１３】


発生器
CPU110
CPU キャッシュ 112
書込収集 バッファ 111
消費器
グラフィクスプロセサ 114
コマンド FIFO 218
コール FIFO 218
890
210
グラフィクス コマンド FIFO
表示リスト 212
主メモリ 112

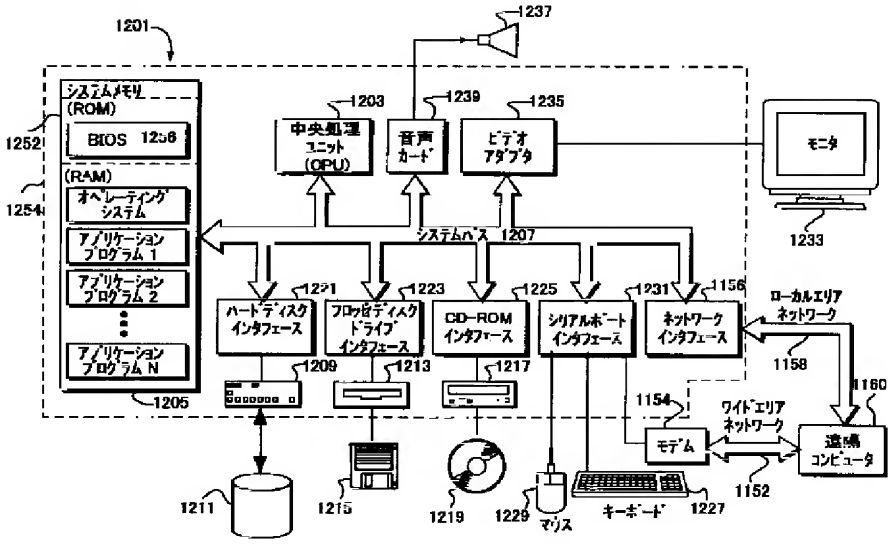
【図１４】


PI_CP_data
CP_PI_data
DFifoAddr
DFifoReq
DFifoAck
FIFO アドレス 計算器
CPイネーブル 820
FIFO ベース 822
FIFO 先頭 824
FIFO HICNT 826
FIFO LOCNT 828
FIFO CNT 830
FIFO ライトポインタ 808
FIFO リードポインタ 804
FIFO 中断点 832
CP ステータス 834
CP イネーブル836
CP クリア 838
DFifoData
DFifoDataV
840
StmCmdData

【図１５】


1201
ホストプラットホーム
(たとえば,
PC,MAC,または
ニンテンドウN64)
1303
エミュレータ
62
ソフトウェア
(ROM
または光ディスク)
命令(ターゲット)
命令取り込み
(ソース)
グラフィクス,
オーディオコマンド(ターゲット)
グラフィクス,
オーディオコマンド
取り込み(ソース)

【図１６】


1201
1237
システムメモリ
(ROM)
1252
BIOS 1256
(RAM)
1254
オペレーティング
システム
アプリケーション
プログラム 1
アプリケーション
プログラム 2
アプリケーション
プログラム N
1205
1203
中央処理
ユニット
(CPU)
1239
音声
カード
1235
ビデオ
アダプタ
モニタ
1233
システムバス 1207
1221
ハードディスク
インタフェース
1223
フロッピディスク
ドライブ
インタフェース
1225
CD-ROM
インタフェース
1231
シリアルポート
インタフェース
1156
ネットワーク
インタフェース
ローカルエリア
ネットワーク
1158
1160
1209
1213
1217
1154
ワイドエリア
ネットワーク
モデム
遠隔
コンピュータ
1152
1211
1215
1219
1229
マウス
キーボード
1227

【外国語明細書】

Method and Apparatus For Buffering Graphics Data In A Graphics System

This application claims the benefit of U.S. Provisional Application No. 60/226,912, filed August 23, 2000, the entire content of which is hereby incorporated by reference in this application.

Field of the Invention

The invention relates to computer graphics, and more particularly to interactive graphics systems such as home video game platforms. Still more particularly, this invention relates to efficient graphics command buffering between a graphics command producer and a graphics command consumer.

Background And Summary Of The Invention

Many of us have seen films containing remarkably realistic dinosaurs, aliens, animated toys and other fanciful creatures. Such animations are made possible by computer graphics. Using such techniques, a computer graphics artist can specify how each object should look and how it should change in appearance over time, and a computer then models the objects and displays them on a display such as your television or a computer screen. The computer takes care of performing the many tasks required to make sure that each part of the displayed image is colored and shaped just right based on the position and orientation of each object in a scene, the direction in which light seems to strike each object, the surface texture of each object, and other factors.

Because computer graphics generation is complex, computer-generated three-dimensional graphics just a few years ago were mostly limited to expensive specialized flight simulators, high-end graphics workstations and supercomputers.

The public saw some of the images generated by these computer systems in movies and expensive television advertisements, but most of us couldn't actually interact with the computers doing the graphics generation. All this has changed with the availability of relatively inexpensive 3D graphics platforms such as, for example, the Nintendo 64® and various 3D graphics cards now available for personal computers. It is now possible to interact with exciting 3D animations and simulations on relatively inexpensive computer graphics systems in your home or office.

A problem graphics system designers confronted in the past was how to efficiently buffer graphics commands between a graphics command producer and a graphics command consumer. Various solutions to this problem were offered. For example, it is well known to provide a buffer memory between a graphics command producer and a graphics command consumer. Often, this buffer memory is connected as part of the graphics command consumer (for example, on board a graphics chip). The graphics command producer writes graphics commands into the buffer memory, and the graphics command consumer reads those graphics commands from the buffer memory. It is typical for such a buffer memory to be structured as a first-in-first-out (FIFO) buffer so that the graphics command consumer reads the graphics command in the same sequence that they were written into the buffer by the graphics command producer.

Placing such a buffer between the producer and the consumer relaxes the degree to which the producer and consumer must be synchronized. The producer can write commands into the buffer at an instantaneous rate that is independent of the instantaneous rate at which the consumer reads commands from the buffer. Even if the consumer suffers a momentary delay in reading from the buffer (e.g., as may occur when the producer asks the consumer to draw large or complex

primitives), the producer will not stall unless/until it fills the buffer and has no more memory space to write new commands. Similarly, momentary delays of the producer in writing new graphics commands into the buffer will not cause the consumer to stall unless the consumer consumes all of the graphics commands in the buffer before the producer has an opportunity to write additional graphics commands.

A potential problem encountered in the past relates to the size of the buffer. Because of limitations on chip size and complexity, it is often not possible to put a very large command buffer memory on the graphics chip. A small sized FIFO buffer in the graphics hardware may not adequately load balance between the producer and the consumer, causing the producer to stall when the consumer renders big primitives. Thus, while significant work has been done in the past, further improvements are possible.

The present invention solves this problem by providing techniques and arrangements that more efficiently buffer graphics commands between a graphics command producer and a graphics command consumer. In accordance with one aspect of the invention, a part of main memory shared between the producer and consumer is allocated to a variable number of variable sized graphics command buffers. The producer can specify the number of buffers and the size of each. Writes to the graphics consumer can be routed to any of the buffers in main memory. A buffer can be attached simultaneously to the consumer and the producer, or different buffers can be attached to the consumer and the producer. In the multi-buffering approach where different buffers are attached to the consumer and the producer, the producer can write to one buffer while the consumer reads from another buffer.

To further decouple the consumer from the producer, the producer and consumer independently maintain their own read and write pointers in accordance with another aspect of the invention. Even though the consumer may not write to the buffer, it nevertheless maintains a write pointer which it uses to keep track of data valid position within the buffer. Similarly, even though the producer may not read from the buffer it is attached to, it maintains a read pointer which it uses to keep track of data valid position within the buffer. The effect of this pointer arrangement is to further decouple the producer from the consumer --reducing the synchronization requirements between the two.

In accordance with another aspect provided by this invention, the producer can write a "call display list" command to a FIFO buffer that directs the consumer to read a string of graphics commands (e.g., a display list) stored elsewhere in memory, and to subsequently return to reading the rest of the buffer. This ability to call an out-of-line graphics command string from a FIFO buffer provides additional flexibility and further decreases synchronization requirements.

In accordance with another aspect of the invention, the graphics command producer can write a graphics command stream to a FIFO buffer that includes a command which automatically redirects succeeding commands to a display list buffer. One way to visualize this is to picture the graphics command producer as a redirectable fire hose that continually produces a stream of graphics commands. The fire hose normal streams the graphics command into a FIFO buffer. However, the producer can include, within the stream, a "Begin Display List" command that causes graphics commands following the command to be written to a display list instead. An "End Display List" command inserted further on in the stream can terminate the display list and redirect the graphics command stream back to the

same (or different) FIFO buffer. This feature has the advantage of allowing the graphics command producer to efficiently create reusable display lists with very low overhead.

In accordance with another aspect provided by this invention, the graphics command producer can insert a break point into any of multiple FIFO buffers. The break point can cause the consumer to interrupt. Such break points can help to synchronize the producer and the consumer when close synchronization is required.

In accordance with yet another aspect provided by this invention, the graphics system includes a producer that outputs graphics commands, a consumer that consumes the graphics commands outputted by the producer, and a storage device coupled between the producer and the consumer. The storage device stores plural variable sized buffers disposed at variable locations within the storage device. Each of the variable sized buffers receives and temporarily stores graphics commands outputted by the producer for delivery to the consumer.

In accordance with a further aspect provided by the invention, the consumer is incapable of writing to at least an active one of the plural buffers, but nevertheless maintains – independently of the producer – a write pointer for at least the active one of the plural buffers. The producer provides a producer read pointer and a producer write pointer associated with a first of the plural buffers, and the consumer independently maintains a consumer read pointer and a consumer write pointer associated with that same buffer. The consumer may increment the consumer read pointer as the consumer reads from an active buffer and suspends reading from the active buffer when the incremented consumer read pointer has a predetermined relationship with a consumer write pointer. The consumer may selectively

increment the consumer write pointer in response to the producer writing to the active buffer.

In accordance with another aspect of the invention, a buffer includes a read command that controls the consumer to consume a set of graphics commands the producer stores elsewhere within the storage device, and to resume consuming graphics commands from the buffer after consuming the graphics commands stored elsewhere. The read command may specify a starting address and a length of a display list. The read command controls the consumer to read the display list of the specified length beginning at the specified starting address.

In accordance with another aspect of the invention, any of the plural buffers may provide either circular or linear first-in-first-out access.

In accordance with another aspect of the invention, any of the plural buffers can be selectively attached to both the producer and the consumer simultaneously – or one of the buffers can be attached to the producer while another buffer is attached to the consumer.

In accordance with still another aspect provided by the invention, the producer allocates the size of each of the plural buffers. Such allocation is provided so that each buffer is capable of storing at least a frame of graphics commands.

In accordance with another aspect of the invention, the producer may write a break point into any of the plural buffers. The consumer may suspend consumption of graphics commands upon encountering the break point.

In accordance with yet another aspect of the invention, each buffer may provide an overflow status indicator indicating when the producer overwrites a location in the buffer.

In accordance with yet another aspect of the invention, a status register or other indicator may indicate the status of at least one of the plural buffers. The status register may indicate, for example:

- producer writer pointer position,
- producer read pointer position,
- consumer write pointer position, and
- consumer read pointer position.

In accordance with yet another aspect provided by this invention, a graphics system includes:

- a storage buffer that receives and temporarily stores graphics commands,
- a producer that writes graphics commands into the buffer, the producer maintaining a producer write pointer and a producer read pointer associated with the buffer, and
- a consumer that consumes graphics commands stored within the buffer, the consumer maintaining a consumer write pointer that is independent of the producer write pointer and a consumer read pointer that is independent of the producer read pointer.

In accordance with yet another aspect of this invention, a graphics system includes a graphics command producer that writes graphics commands into a buffer based on a producer write pointer, and a graphics commands consumer that reads graphics commands from the buffer based on a consumer read pointer. In accordance with this aspect of the invention, the consumer write pointer is independently maintained by the consumer and indicates the extent of valid data the producer has written into the buffer. The consumer ceases to consume graphics

commands from the buffer upon the consumer read pointer having a predetermined relationship to the consumer write pointer.

In accordance with yet another aspect provided by this invention, an interactive graphics system includes a processor module executing an application, a graphics processor module, and at least one memory coupled to the processor module and to the graphics processor module. The method of controlling the flow of graphics commands between the processor module and the graphics processor module comprises:

- dynamically establishing, under control of the application, a variable number of FIFO buffers in the memory, the application specifying the size of each of the FIFO buffers,
- the application controlling the processor module to write graphics commands into at least a first of the plurality of FIFO buffers, and
- the application sending graphics commands to the graphics processor module that control the graphics processor module to read graphics commands from the first FIFO buffer.

The processor module may provide a processor module read pointer and processor module write pointer associated with the first of plurality of buffers. The graphics processor module may independently maintain a graphics processor module read pointer and a graphics processor module write pointer associated with the first buffer. The graphics processor module may increment the graphics processor read pointer each time the graphics processor module reads from the first buffer, and may suspend reading from the first buffer when the graphics processor module read pointer has a predetermined relationship with the graphics processor module write pointer. Graphics processor module may selectively auto increment

the graphics processor write pointer in response to the processor writing to the first buffer.

In accordance with yet another aspect of the invention, a method of controlling the flow of graphics data comprises:

- writing graphics data into plural variable sized FIFO buffers each having plural storage locations,
- setting a break point associated with at least one of the plural storage locations,
- reading graphics data from the plural buffers in a predetermined order,
- temporarily suspending the reading step upon encountering the at least one location associated with the break point, and generating an interrupt, and
- resuming the reading step in response to receipt of a clear interrupt command.

In accordance with yet another aspect provided by this invention, a graphics system includes:

- a storage device that receives and temporarily stores graphics commands,
- a producer that writes commands into a buffer within the storage device, the commands including a first set of graphics commands and a read command referring to a second set of graphics commands stored elsewhere in the storage device, and
- a consumer that consumes the first set of graphics commands stored within the buffer and, in response to encountering the read

command, consumes the second set of graphics commands and subsequently consumes additional commands from the buffer.

Brief Description Of The Drawings

These and other features and advantages provided by the invention will be better and more completely understood by referring to the following detailed description of presently preferred embodiments in conjunction with the drawings, of which:

Figure 1 is an overall view of an example interactive computer graphics system;

Figure 2 is a block diagram of the Figure 1 example computer graphics system;

Figure 3 is a block diagram of the example graphics and audio processor shown in Figure 2;

Figure 4 is a block diagram of the example 3D graphics processor shown in Figure 3;

Figure 5 is an example logical flow diagram of the Figure 4 graphics and audio processor;

Figure 6 shows example multi-buffering;

Figure 7 shows example independent consumer and producer read and write pointers;

Figures 8A and 8B show, respectively, example empty and full buffer conditions;

Figure 9 shows an example call of a display list from an FIFO buffer;

Figures 10A-10C show example display list creation; and

Figure 11 shows an example FIFO manager implementation; and

Figures 12A and 12B show example alternative compatible implementations.

Detailed Description Of Example Embodiments Of The Invention

Figure 1 shows an example interactive 3D computer graphics system 50. System 50 can be used to play interactive 3D video games with interesting stereo sound. It can also be used for a variety of other applications.

In this example, system 50 is capable of processing, interactively in real time, a digital representation or model of a three-dimensional world. System 50 can display some or all of the world from any arbitrary viewpoint. For example, system 50 can interactively change the viewpoint in response to real time inputs from handheld controllers 52a, 52b or other input devices. This allows the game player to see the world through the eyes of someone within or outside of the world. System 50 can be used for applications that do not require real time 3D interactive display (e.g., 2D display generation and/or non-interactive display), but the capability of displaying quality 3D images very quickly can be used to create very realistic and exciting game play or other graphical interactions.

To play a video game or other application using system 50, the user first connects a main unit 54 to his or her color television set 56 or other display device by connecting a cable 58 between the two. Main unit 54 produces both video signals and audio signals for controlling color television set 56. The video signals are what controls the images displayed on the television screen 59, and the audio signals are played back as sound through television stereo loudspeakers 61L, 61R.

The user also needs to connect main unit 54 to a power source. This power source may be a conventional AC adapter (not shown) that plugs into a standard home electrical wall socket and converts the house current into a lower DC voltage signal suitable for powering the main unit 54. Batteries could be used in other implementations.

The user may use hand controllers 52a, 52b to control main unit 54. Controls 60 can be used, for example, to specify the direction (up or down, left or right, closer or further away) that a character displayed on television 56 should move within a 3D world. Controls 60 also provide input for other applications (e.g., menu selection, pointer/cursor control, etc.). Controllers 52 can take a variety of forms. In this example, controllers 52 shown each include controls 60 such as joysticks, push buttons and/or directional switches. Controllers 52 may be connected to main unit 54 by cables or wirelessly via electromagnetic (e.g., radio or infrared) waves.

To play an application such as a game, the user selects an appropriate storage medium 62 storing the video game or other application he or she wants to play, and inserts that storage medium into a slot 64 in main unit 54. Storage medium 62 may, for example, be a specially encoded and/or encrypted optical and/or magnetic disk. The user may operate a power switch 66 to turn on main unit 54 and cause the main unit to begin running the video game or other application based on the software stored in the storage medium 62. The user may operate controllers 52 to provide inputs to main unit 54. For example, operating a control 60 may cause the game or other application to start. Moving other controls 60 can cause animated characters to move in different directions or change the user's point of view in a 3D world. Depending upon the particular software stored within the storage medium 62, the

various controls 60 on the controller 52 can perform different functions at different times.

Example Electronics of Overall System

Figure 2 shows a block diagram of example components of system 50. The primary components include:

- a main processor (CPU) 110,
- a main memory 112, and
- a graphics and audio processor 114.

In this example, main processor 110 (e.g., an enhanced IBM Power PC 750) receives inputs from handheld controllers 108 (and/or other input devices) via graphics and audio processor 114. Main processor 110 interactively responds to user inputs, and executes a video game or other program supplied, for example, by external storage media 62 via a mass storage access device 106 such as an optical disk drive. As one example, in the context of video game play, main processor 110 can perform collision detection and animation processing in addition to a variety of interactive and control functions.

In this example, main processor 110 generates 3D graphics and audio commands and sends them to graphics and audio processor 114. The graphics and audio processor 114 processes these commands to generate interesting visual images on display 59 and interesting stereo sound on stereo loudspeakers 61R, 61L or other suitable sound-generating devices.

Example system 50 includes a video encoder 120 that receives image signals from graphics and audio processor 114 and converts the image signals into analog and/or digital video signals suitable for display on a standard display device such as

a computer monitor or home color television set 56. System 50 also includes an audio codec (compressor/decompressor) 122 that compresses and decompresses digitized audio signals and may also convert between digital and analog audio signaling formats as needed. Audio codec 122 can receive audio inputs via a buffer 124 and provide them to graphics and audio processor 114 for processing (e.g., mixing with other audio signals the processor generates and/or receives via a streaming audio output of mass storage access device 106). Graphics and audio processor 114 in this example can store audio related information in an audio memory 126 that is available for audio tasks. Graphics and audio processor 114 provides the resulting audio output signals to audio codec 122 for decompression and conversion to analog signals (e.g., via buffer amplifiers 128L, 128R) so they can be reproduced by loudspeakers 61L, 61R.

Graphics and audio processor 114 has the ability to communicate with various additional devices that may be present within system 50. For example, a parallel digital bus 130 may be used to communicate with mass storage access device 106 and/or other components. A serial peripheral bus 132 may communicate with a variety of peripheral or other devices including, for example:

- a programmable read-only memory and/or real time clock 134,
- a modem 136 or other networking interface (which may in turn connect system 50 to a telecommunications network 138 such as the Internet or other digital network from/to which program instructions and/or data can be downloaded or uploaded), and
- flash memory 140.

A further external serial bus 142 may be used to communicate with additional expansion memory 144 (e.g., a memory card) or other devices. Connectors may be used to connect various devices to busses 130, 132, 142.

Example Graphics And Audio Processor

Figure 3 is a block diagram of an example graphics and audio processor 114. Graphics and audio processor 114 in one example may be a single-chip ASIC (application specific integrated circuit). In this example, graphics and audio processor 114 includes:

- a processor interface 150,
- a memory interface/controller 152,
- a 3D graphics processor 154,
- an audio digital signal processor (DSP) 156,
- an audio memory interface 158,
- an audio interface and mixer 160,
- a peripheral controller 162, and
- a display controller 164.

3D graphics processor 154 performs graphics processing tasks. Audio digital signal processor 156 performs audio processing tasks. Display controller 164 accesses image information from main memory 112 and provides it to video encoder 120 for display on display device 56. Audio interface and mixer 160 interfaces with audio codec 122, and can also mix audio from different sources (e.g., streaming audio from mass storage access device 106, the output of audio DSP 156, and external audio input received via audio codec 122). Processor interface 150

provides a data and control interface between main processor 110 and graphics and audio processor 114.

Memory interface 152 provides a data and control interface between graphics and audio processor 114 and memory 112. In this example, main processor 110 accesses main memory 112 via processor interface 150 and memory interface 152 that are part of graphics and audio processor 114. Peripheral controller 162 provides a data and control interface between graphics and audio processor 114 and the various peripherals mentioned above. Audio memory interface 158 provides an interface with audio memory 126.

Example Graphics Pipeline

Figure 4 shows a more detailed view of an example 3D graphics processor 154. 3D graphics processor 154 includes, among other things, a command processor 200 and a 3D graphics pipeline 180. Main processor 110 communicates streams of data (e.g., graphics command streams and display lists) to command processor 200. Main processor 110 has a two-level cache 115 to minimize memory latency, and also has a write-gathering buffer 111 for uncached data streams targeted for the graphics and audio processor 114. The write-gathering buffer 111 collects partial cache lines into full cache lines and sends the data out to the graphics and audio processor 114 one cache line at a time for maximum bus usage.

Command processor 200 receives display commands from main processor 110 and parses them -- obtaining any additional data necessary to process them from shared memory 112. The command processor 200 provides a stream of vertex commands to graphics pipeline 180 for 2D and/or 3D processing and rendering. Graphics pipeline 180 generates images based on these commands. The resulting

image information may be transferred to main memory 112 for access by display controller/video interface unit 164 -- which displays the frame buffer output of pipeline 180 on display 56.

Figure 5 is a logical flow diagram of graphics processor 154. Main processor 110 may store graphics command streams 210, display lists 212 and vertex arrays 214 in main memory 112, and pass pointers to command processor 200 via bus interface 150. The main processor 110 stores graphics commands in one or more graphics first-in-first-out (FIFO) buffers 210 it allocates in main memory 110. The command processor 200 fetches:

- command streams from main memory 112 via an on-chip FIFO memory buffer 216 that receives and buffers the graphics commands for synchronization/flow control and load balancing,
- display lists 212 from main memory 112 via an on-chip call FIFO memory buffer 218, and
- vertex attributes from the command stream and/or from vertex arrays 214 in main memory 112 via a vertex cache 220.

Command processor 200 performs command processing operations 200a that convert attribute types to floating point format, and pass the resulting complete vertex polygon data to graphics pipeline 180 for rendering/rasterization. A programmable memory arbitration circuitry 130 (see Figure 4) arbitrates access to shared main memory 112 between graphics pipeline 180, command processor 200 and display controller/video interface unit 164.

Figure 4 shows that graphics pipeline 180 may include:

- a transform unit 300,

- a setup/rasterizer 400,
- a texture unit 500,
- a texture environment unit 600, and
- a pixel engine 700.

Transform unit 300 performs a variety of 2D and 3D transform and other operations 300a (see Figure 5). Transform unit 300 may include one or more matrix memories 300b for storing matrices used in transformation processing 300a. Transform unit 300 transforms incoming geometry per vertex from object space to screen space; and transforms incoming texture coordinates and computes projective texture coordinates (300c). Transform unit 300 may also perform polygon clipping/culling 300d. Lighting processing 300e also performed by transform unit 300b provides per vertex lighting computations for up to eight independent lights in one example embodiment. Transform unit 300 can also perform texture coordinate generation (300c) for embossed type bump mapping effects, as well as polygon clipping/culling operations (300d).

Setup/rasterizer 400 includes a setup unit which receives vertex data from transform unit 300 and sends triangle setup information to one or more rasterizer units (400b) performing edge rasterization, texture coordinate rasterization and color rasterization.

Texture unit 500 (which may include an on-chip texture memory (TMEM) 502) performs various tasks related to texturing including for example:

- retrieving textures 504 from main memory 112,
- texture processing (500a) including, for example, multi-texture handling, post-cache texture decompression, texture filtering, embossing, shadows

and lighting through the use of projective textures, and BLIT with alpha transparency and depth,

- bump map processing for computing texture coordinate displacements for bump mapping, pseudo texture and texture tiling effects (500b), and
- indirect texture processing (500c).

Texture unit 500 outputs filtered texture values to the texture environment unit 600 for texture environment processing (600a). Texture environment unit 600 blends polygon and texture color/alpha/depth, and can also perform texture fog processing (600b) to achieve inverse range based fog effects. Texture environment unit 600 can provide multiple stages to perform a variety of other interesting environment-related functions based for example on color/alpha modulation, embossing, detail texturing, texture swapping, clamping, and depth blending..

Pixel engine 700 performs depth (z) compare (700a) and pixel blending (700b). In this example, pixel engine 700 stores data into an embedded (on-chip) frame buffer memory 702. Graphics pipeline 180 may include one or more embedded DRAM memories 702 to store frame buffer and/or texture information locally. Z compares 700a' can also be performed at an earlier stage in the graphics pipeline 180 depending on the rendering mode currently in effect (e.g., z compares can be performed earlier if alpha blending is not required). The pixel engine 700 includes a copy operation 700c that periodically writes on-chip frame buffer 702 to main memory 112 for access by display/video interface unit 164. This copy operation 700c can also be used to copy embedded frame buffer 702 contents to textures in the main memory 112 for dynamic texture synthesis effects. Anti-aliasing and other filtering can be performed during the copy-out operation. The frame buffer output of graphics pipeline 180 (which is ultimately stored in main

memory 112) is read each frame by display/video interface unit 164. Display controller/video interface 164 provides digital RGB pixel values for display on display 102.

FIFO Buffers Allocated In Shared Memory

In this example, the command FIFO buffer 216 (which may be a small dual ported RAM streaming buffer) on board the graphics and audio processor 114 is too small, by itself, to do a good job of load balancing between the processor 110 and the graphics pipeline 180. This may result in the processor 110 becoming stalled when the graphics and audio processor 114 is rendering big primitives. To remedy this problem, we use part of the main memory 112 shared between processor 110 and graphics and audio processor 114 as a command FIFO buffer 210. The use of buffers 210 allows the main processor 110 and the graphics processor 114 to operate in parallel at close to their peak rates.

There are (at least) two methods of using buffers 210 to achieve parallelism: immediate mode and multi-buffer mode. When a single buffer 210 is attached to both the main processor 110 and the graphics processor 114, the system 50 is operating in the immediate mode. As the main processor 110 writes graphics commands to the buffer 210, the graphics processor 114 processes them in order. Hardware support provides flow control logic to prevent writes from overrunning reads and to wrap the read and write pointers of the buffer 210 back to the first address to provide circular buffer operation.

In the preferred embodiment, it is also possible to connect one buffer 210 to the main processor 110 while the graphics and audio processor 114 is reading from a different buffer 210(1) in a multi-buffered mode. In this case, the buffers 210(1),

210(2) are managed more like buffers than traditional FIFOs since there are no simultaneous reads and writes to any particular buffer 210. Multi-buffer mode may be used, for example, if dynamic memory management of the buffers is desirable.

Figure 6 shows how a portion of shared memory 112 can be allocated to provide multiple FIFO command buffers 210(1), 210(2), ..., 210(n) to buffer graphics (and audio) commands between the producer 110 and the consumer 114. In the example shown in Figure 6, each of buffers 210 receives graphics (and/or audio) commands from main processor 110, and provides those commands to graphics and audio processor 114. Main processor 110 allocates portions of main memory 112 for use as these buffers 210. A buffer data structure describing a region of main memory can be allocated by an application running on main processor 110.

Main processor 110 writes graphics commands into the buffers using a write pointer 802. Graphics and audio processor 114 reads commands from buffers 210 using a read pointer 804. Write pointer 802 and read pointer 804 can point to the same or different buffers. In this way, the same buffer 210 may be "attached" to both the main processor 110 and the graphics and audio processor 114 simultaneously – or different buffers may be attached to the producer and consumer at different times.

In the multi-buffering example shown in Figure 6, the main processor 110 and the graphics and audio processor 114 don't necessarily agree on where "the" FIFO buffer 210 is located. In the example shown, the main processor 110 is using buffer 210(2) as its current buffer for writing graphics commands to, whereas the graphics and audio processor 114 uses a different buffer 210(1) as its current buffer for obtaining graphics commands. Buffers 210 can be dynamically attached to main processor 110, graphics and audio processor 114, or both. When a buffer is

attached to the main processor 110, the main processor will write graphics commands into the buffer 210. In the example embodiment, there is always one and only one buffer 210 attached to main processor 110 at any one time. When a buffer 210 is attached to the graphics processor 114, the graphics processor will read and process graphics commands from the attached buffer 210. Only one buffer 210 can be attached to the graphics processor 114 at any one time in this example.

Independent Consumer and Producer Read and Write Pointers

Even though main processor 110 acting as graphics command producer does not need to read from the buffer 210(2) to which it is attached, it nevertheless maintains a producer read pointer 806 in this Figure 6 example. Similarly, even though the graphics and audio processor 114 acts as a consumer of graphics commands and therefore does not need to write to the buffer 210(1) to which it is attached, it nevertheless maintains a consumer write pointer 808 in the Figure 6 example. These additional pointers 806, 808 allow the producer and consumer to independently maintain the respective buffer 210 to which it is attached.

The additional pointer 806 maintained by main processor 110 and the additional pointer 808 maintained by graphics and audio processor 114 are used to provide overlap detection. These extra pointers indicate where valid data exists within the buffer 210. For example, the main processor 110 may treat the buffer 210(2) to which it is attached as a circular buffer, and "wrap" its write pointer around to the "beginning" of the buffer 810 once it reaches the "end" of the buffer 812. However, once the producer write pointer 802 encounters the producer read pointer 806, it will cease writing to attached buffer 210(2) to avoid overwriting valid, previously written data that the graphics and audio processor 114 has not yet read. Similarly, the graphics and audio processor consumer 114 may continue to

increment its read pointer 804 as it progressively reads graphics instructions from its attached buffer 210(1), but will cease this incrementing procedure when the read pointer 804 encounters the write pointer 808 – since the consumer is using the write pointer as indicating the last valid data within the buffer 210(1).

Pointers 802, 804, 806, and 808 can point to any location within buffers 210. Valid data may thus exist anywhere within these buffers – not necessarily at the beginning or at the end of the buffer. In fact, if buffers 210 are operated in a circular mode, there is no concept of "beginning" or "end" since the end of the buffer wraps around to the beginning and the buffer is therefore a logically continuous loop.

Figure 7 provides a simplified explanation of the independent consumer and producer read and write pointers. In the Figure 7 example, consumer 114 uses an auto-incrementing read pointer 804 to read graphics commands from the buffer 210(1) to which it is attached. Consumer 114 also maintains a consumer write pointer 808 that points to the last valid graphics command within buffer 210(1). In this example, consumer 114 will continue to read graphics commands from buffer 210(1), and increment its read pointer 804 after each graphics command read, until the read pointer points to the same location that the write pointer points to (see Figure 8A). When the consumer 114 has incremented its read pointer 804 so that it points to the location adjacent the one that the write pointer 808 points to, the consumer "knows" that it has read all of the valid graphics commands from buffer 210(1) and has thus emptied the buffer. This condition indicates that the consumer 114 either needs to wait for more graphics commands from producer 110 (if the buffer 210(1) is also attached to the producer simultaneously), or it needs direction

as to a different buffer 210 it should begin reading from (if multi-buffering is in effect).

Similarly, the producer 110 may continue to write graphics commands into its attached buffer 210(2) and similarly continues to auto-increment its producer write pointer 802 until the write pointer points to the location in the buffer that is just before the location the producer read pointer 806 points to (see Figure 8B). In this example, coincidence (actually, close proximity) between the write pointer 802 and read pointer 806 indicates that the buffer 210(2) is full. If multi-buffering is in effect, producer 110 may at this point cease writing to buffer 210(2) and "save" (close) it, instruct the consumer 114 to read (now or later) the contents of that "closed" buffer, and begin writing additional graphics commands to yet another buffer 210 it can allocate within main memory 112. If the producer 110 and consumer 114 are attached to the same buffer 210, then the producer may need to wait until the consumer reads some commands before writing any more commands to the buffer. As explained below, to avoid frequent context switching, the preferred embodiment can provides a programmable hysteresis effect that requires the buffer to be emptied by a certain amount before the producer 110 is allowed to resume writing to the buffer, and requires the buffer to be filled by a certain amount before the consumer is allowed to resume reading from the buffer.

In the preferred embodiment, the main processor 110 writes graphics commands to the buffer 210 to which it is attached in 32-byte transfers. Main processor 110 provides a write-gathering buffer/function 111 (see Figure 4) that automatically packs graphics commands into 32-byte words. Graphics processor 114 reads graphics commands from the buffer 210 to which it is attached in 32-byte transfers.

Call Display List From FIFO Buffer

Figure 9 shows an example technique provided by the preferred example embodiment whereby an entry in a FIFO buffer 210 can call a display list – almost as if it were a function call. In this example, a command 890 is inserted into the graphics command FIFO 210 that calls a display list 212 stored elsewhere in memory. Upon encountering this command 890, the graphics processor 114 temporarily ceases reading graphics commands from FIFO buffer 210 and instead begins reading commands from a display list 212 stored elsewhere in main memory 112. Upon reaching the end of the display list 212, the graphics processor 114 returns to read the next sequential command from the graphics FIFO 210. This technique is quire useful in allowing multiple frames to call the same display list 212 (e.g., to render geometry which remains static from frame to frame) without requiring the main processor 112 to rewrite the display list for each frame.

Figures 10A through 10C show how main processor 110 can automatically create a display list 212 by writing to a graphics command FIFO 210. As shown in Figure 10A, main processor 110 begins by writing a graphics command stream to a graphics command FIFO 210 it allocates in main memory 112. At any point in this writing process, the main processor 110 can insert a "Begin Display List" command 890 into the FIFO buffer 210 that causes further writes from the main processor to be directed to a display list 212. Figure 10C shows that once main processor 110 is finished writing display list 212, it may issue an "End Display List" command that has the effect of automatically terminating the display list and redirecting the main processor command stream output back to FIFO buffer 210. One can visualize main processor 110 providing a redirectable "fire hose" command stream output that can gush graphics commands into FIFO buffer 210, display list 212, and back to the

same or different FIFO buffer 212. The display lists 212 created in this manner can remain in memory 112 and reused for parts of images that remain static over several frames or frame portions.

Example Implementation Details

A processor to graphics interface unit portion 202 of the graphics and audio processor 114 command processor 200 contains the control logic for managing the FIFO buffers 210 in main memory 112. Figure 11 shows an example implementation. In the example shown, all CPU 110 writes to the graphics and audio processor 114 will be routed to the main memory 112. There are two registers that define the portion of the main memory 112 that has been allocated to the graphics FIFO 210 attached to the graphics and audio processor 114:

the FIFO BASE register 822, and

the FIFO TOP register 824.

The FIFO_BASE register 822 defines the base address of the FIFO 210. The FIFO_TOP register 824 defines the last address in the FIFO.

Command processor 200 keeps track of the read and write pointers for FIFO 210 in hardware. Since all data written into the FIFO are cache line sized, there is no need to keep track of valid bytes. The write pointer 808 is incremented by 32 bytes every a cache line is written to an address that is between FIFO_BASE and FIFO_TOP (5LSBs are 0). Reading of the FIFO 210 is also performed one cache line at a time. The read pointer is incremented by 32 after a cache line has been read.

Initially, read pointer 804 and write pointer 808 are initialized to point to the same location, which means the FIFO is empty (see Figure 8A). The FIFO full

condition is (read pointer – 1) = (write pointer) (see Figure 8B). Write pointer 808 wraps around to the FIFO_BASE 204(2) address after it reaches FIFO_TOP. The read pointer 804 also wraps around when it reaches FIFO_TOP 824. The read pointer 804 is controlled by the hardware to make sure it doesn't get ahead of the write pointer 808, even in the wrap around cases. The application running on processor 110 makes sure that the write pointer 808 doesn't surpass the read pointer 804 after wrapping around.

Data from two (or more) different frames can be resident in the same FIFO 210. A break point mechanism can be used to prevent the command processor 200 from executing the second frame before the first frame can be copied out of the embedded DRAM 702. When FIFO break point (register) 832 is enabled, command processor 200 will not read past the CP_FIFO_BRK register. The CPU 100 can program this register 832 at the end of a frame. CPU 110 has to flush the write-buffer on the graphics and audio processor 114 and then read the FIFO write pointer 808. It then writes the value into the FIFO break register 832 and enables the break point.

If the size of the FIFO 210 is big enough to hold all the data sent in one frame, then the FIFO full condition shown in Figure 8B will never occur. However, this could mean allocating 2 to 4 Mbytes of main memory 112 for the FIFO buffer 210. Some application developers might not want to use that much memory for FIFO 210. In that case, the application should implement a flow control technique. Registers 826, 828 can be used to provide such flow control. Flow control is done in the example embodiment by having graphics and audio processor 114 generate an interrupt back to the CPU 110 when the number of cache lines in the main memory 110 surpasses FIFO_HICNT 826. The processor 110 will take the interrupt and

spin or do other non-graphical tasks, until the number of cache-lines in the FIFO is less than a FIFO_LOCNT 828. The reason for providing such a hysteresis effect is that interrupt overhead is high and one does not want to bounce in and out of the interrupt routine just by checking that the contents of the FIFO 210 has gone below the "high water mark". Interrupts can also be generated when the FIFO count goes below the LOCNT 828. This way, the application can perform other tasks and return when interrupted.

Example FIFO Buffer Allocation

In the preferred embodiment, the graphics API declares a static FXFifoObj structure internally. This structure is initialized when GXInit is called:

GXFifoObj* GXInit (void* base, u32 size);

The FIFO base pointer is aligned to 32b in the preferred embodiment. The application is responsible for allocating the memory for the FIFO. The size parameter for allocation is the size of the FIFO in bytes (the minimum FIFO size is 64KB, and size is a multiple of 32B). By default, GXInit sets up the FIFO for immediate mode graphics; that is: both the CPU 110 and graphics processor 114 are attached to the FIFO, the read and write pointers are initialized to the base pointer, and high and low water marks are enabled. GXInit returns a pointer to the initialized GXFifoObj to the application.

If the application wants to operate in multi-buffered mode, then additional FIFOs must be allocated. Any number of such additional FIFO buffers 210 can be allocated. The application allocates the memory for each additional FIFO and initializes a GXFifoObj as well. The following example functions can be used to initialize the GXFifoObj:

```
void GXInitFifoBase(
        GXFifoObj*   fifo,
        void*        base,
        u32          size);
void GXInitFifoPtrs(
        GXFifoObj*   fifo
        void*        read_ptr,
        void*        write_ptr );
void GXInitFifoLimits(
        GXFifoObj*   fifo,
        u32          hi_water_mark,
        u32          lo_water_mark );
```

Normally, the application only needs to initialize the FIFO read and write pointers to the base address of the FIFO. Once initialized, the system hardware will control the read and write pointers automatically.

Attaching and Saving FIFOs

Once a FIFO has been initialized, it can be attached to the CPU 110 or the graphics processor 114 or both. Only one FIFO may be attached to either the CPU 110 or graphics processor 114 at the same time. Once a FIFO is attached to the CPU 110, the CPU may issue GX commands to the FIFO. When a FIFO is attached to the graphics processor 114, it will be enabled to read graphics commands from the FIFO. The following example functions attach FIFOs:

```
void GXSetCPUFifo( GXFifoObj* fifo );
void GXSetGPFifo( GXFifoObj* fifo );
GXFifoObj* GXGetCPUFifo ( void ) ;
GXFifoObj* GXGetGPFifo ( void );
```

One may also inquire which FIFO objects are currently attached with these example functions:

```
GXFifoObj* GXGetCPUFifo ( void );
GXFifoObj* GXGetGPFifo ( void );
```

When in multi-buffer mode, and the CPU 110 is finished writing GX commands, the FIFO should be "saved" before switching to a new FIFO. The following example function "saves" the CPU FIFO:

```
void GXSaveCPUFifo ( FXFifoObj* fifo );
```

When a FIFO is saved, the CPU write-gather buffer 111 is flushed to make sure all graphics commands are written to main memory 112. In addition, the current FIFO read and write pointers are stored in the GXFifoObj structure.

Notice that there is no save function for the graphics processor 114. Once a graphics processor is attached, graphics commands will continue to be read until either:

- the FIFO is empty,
- a FIFO breakpoint is encountered, or
- the GP is pre-empted.

FIFO Status

The following example functions can be used to read the status of a FIFO and the GP:

```
void GXGetFifoStatus(
        GXFifoObj*   fifo,
        GXBool*      overhi,
        GXBool*      underlo,
        u32*         fifo_cnt,
        GXBool*      cpu_write,
        GXBool*      gp_read,
        GXBool*      fifowrap );
```

```
void GXGetGPStatus(
        GXBool*     overhi,
        GXBool*     underlow,
        GXBool*     readIdle,
        GXBool*     cmdIdle,
        GXBool*     brkpt );
```

GXGetFifoStatus gets the status of a specific FIFO. If the FIFO is currently attached to the CPU 110, the parameter cpu_write will be GX_TRUE. When the FIFO is currently attached to the graphics processor 114, the parameter gp_read will be GX_TRUE. When a FIFO is attached to either the CPU 110 or the graphics processor 114, the status will be read directly from the hardware's state. If the FIFO is not attached, the status will be read from the GXFifoObj. GXGetFifoStatus reports whether the specified FIFO has over flowed or has enough room to be written to. In general, the hardware cannot detect when a FIFO overflows, i.e., when the amount of data exceeds the size of the FIFO.

Although there is no general way to detect FIFO overflows, the hardware can detect when the CPU write pointer reaches the top of the FIFO. If this condition has occurred, the "fifowrap" argument will return GX_TRUE. The "fifowrap" argument can be used to detect FIFO overflows if the CPU's write pointer is always initialized to the base of the FIFO. "fifowrap" is set if the FIFO is currently attached to the CPU 110.

GXGetGPStatus can be used to get the status of the graphics processor 114 (regardless of the FIFO that attached to it). The minimum requirement to meet before attaching a new graphics processor FIFO is to wait for the graphics processor 114 to be idle (but additional constraints may also exist). The underlow and overhi

statuses indicate where the write pointer is, relative to the high and low water marks.

Example FIFO Flow Control

When a FIFO is attached to both the CPU and GP (immediate mode), care must be taken so that the CPU 110 stops writing commands when the FIFO is too full. A "high water mark" defines how full the FIFO can get before graphics commands will no longer be written to the FIFO. In the preferred embodiment, there may be up to 16KB of buffered graphics commands in the CPU, so it is recommended to set the high water mark to the (FIFO size – 16KB).

When the high water mark is encountered, the program will be suspended, but other interrupt-driven tasks such as audio will still be service. The programmer may also wish to specify which particular thread in a multi-threaded program should be suspended.

A "low water mark" defines how empty the FIFO must get after reaching a "high water mark" before the program (or thread) is allowed to continue. The low water mark is recommended to be set to (FIFO size / 2). The low water mark prevents frequent context switching of the program, since it does not need to poll some register or constantly receive overflow interrupts when the amount of new command data stays close to the high water mark.

When in multi-buffered mode, the high and low water marks are disabled. When a FIFO is attached to the CPU 110, and the CPU writes more commands than the FIFO will hold, the write pointer will be wrapped from the last address back to the base address. Previous graphics commands in the FIFO will be overwritten. It is possible to detect when the write pointer wraps over the top of the FIFO (which

indicates an overflow only if the FIFO write pointer was initialized to the base of the FIFO before commands were sent). See GXGetFifoStatus above.

In order to prevent FIFO (buffer) overflow in multi-buffered mode, a software-based checking scheme may be used. The program running on main processor 110 should keep its own counter of the buffer size, and before any group of commands is added to the buffer, the program may check and see if there is room. If room is available, the size of the group may be added to the buffer size. If room is not available, the buffer may be flushed and a new one allocated.

Using Display List Calls

To call a display list from a FIFO buffer 210 in the preferred embodiment, the application first allocates space in memory in which to store the display list. Once the memory area has been set up, the application can then call for example:

```
void GXBeginDisplayList (
        void        *list
        u32         size);
```

Where the "list" argument is the starting address for where the display list will be stored and the "size" argument indicates the number of bytes available in the allocated space for writing display list commands to allow the system to check for overflow.

Once "GXBeginDisplayList" has been called, further GX commands are written to the display list instead of to the normal command FIFO. The "GXEndDisplayList" command signals the end of the display list, and it returns the command steam to the FIFO to which it had been directed previously. The

"GXEndDisplayList" command also returns the actual size of the created display list as a multiple of 32 bytes in the example embodiment.

In the example embodiment, display lists cannot be nested. This means that once a GXBeginDisplayList has been issued, it is illegal to issue either another GXBeginDisplayLit or a GXCallDisplayList command until a GXEndDisplayList command comes along. However, in alternate embodiments it would be possible to provide display list nesting to any desired nesting level.

Example Graphics FIFO Functions

The following example functions provide management of the graphics FIFO:

GXSetFifoBase:

Argument:

```
u32      BasePtr;      //Set base address of fifo in main memory.
u32      Size;         //Size of the fifo in bytes. (a 32 bytes
                       multiple).
GXBool   Set Defaults  //Setup default fifo state.
```

Sets the graphics fifo limits. This function is called at initialization time. The fifo address can not be changed unless the graphics pipe is flushed. If SetDefault flag is set, then the fifo is reset (i.e., read/write pointers at fifo base) and interrupts are disabled. By default, the high water mark is set to 2/3 of the size and the low water mark is set to 1/3 of the size.

GXSetFifoLimits:

Argument

```
u32      HiWaterMark;   //Hi-water mark for the fifo.
u32      LoWaterMark;   //Low water mark.
u32      RdBreakMark;   //Read pointer break point.
```

This function sets the fifo limits. When the read pointer goes below low water mark or when write pointer goes above high water mark, the graphics hardware will interrupt the CPU. The RdBreakMark is used for setting read pointer break point.

GXSetInterrupts:

Argument

```
GXBool  Underflow;   //Enable/Disable low water mark interrupt.
GXBool  Overflow;    //Enable/Disable high water mark interrupt.
GXBool  BreakPoint;  //Enable/Disable fifo read break point.
```

Enables or disables fifo related interrupts. The BreakPoint is a feature than can be used to halt fifo reads by the CP while a previous frame is still being copied.

GXClearInterrupts:

Argument:

```
GXBool  Underflow;   //Clear low water mark interrupt
GXBool  Overflow;    //Clear high water mark interrupt.
GXBool  BreakPoint   //Clear fifo read break point.
```

Clears a pending interrupt.

GXSetFifoPtrs:

Argument:

```
u32     WritePtr;    //Sets write pointer for fifo.
u32     ReadPtr;     //Sets read pointer.
```

Sets fifo read and write pointers. These pointers are maintained by the hardware. This function will override the hardware values (e.g., for display list compilation).

GXGetFifoStatus:

Argument:

```
GXBool  *UnderFlow;   //Fifo count is below low water mark.
GXBool  *OverFlow;    //Fifo count is above high water mark.
GXBool  *BreakPoint;  //Fifo read pointer is at break point.
u32     *FifoCount;   //Number of cachelines (32 bytes) in Fifo.
```

Returns fifo status and count.

Example Display List Functions

A display list is an array of pre-compiled commands and data for the graphics pipe. The following example commands are inserted into a FIFO buffer 210 to manipulate display lists.

GXBeginDisplayList:

Argument:

```
void*   BasePtr;   //Address of a buffer in for storing display
                   list data.
u32     nBytes;    //Size of the buffer.
```

This function creates and starts a display list. The API is put in display list mode. All API functions, except any of the display list functions, following this call until EndDisplayList, send their data and commands to the display list buffer instead of graphics pipe. A display list can not be nested in this example, i.e., no display list functions can be called between a BeginDisplayList and EndDisplayList. The memory for the display list is allocated by the application.

GXEndDisplayList:

Argument:

None.

Return:

u32 nBytes //Number of bytes used for the display list.

This function ends currently opened display object and puts the system back in immediate mode.

GXCallDisplayList:

Argument:

```
void*   BasePtr;   //Address of a buffer in for storing display list data.
u32     nBytes;    //Size of the buffer
```

This function executes the display list.

Example Register Formats:

The following table shows example registers in the command processor 200 that are addressable by CPU 110:

Register Name	Bit Fields: Description
CP_STATUS Register 834	0: FIFO overflow (fifo_count > FIFO_HICNT)
	1: FIFO underflow (fifo_count < FIFO_LOCNT)
	2: FIFO read unit idle
	3: CP idle
	4: FIFO reach break point (cleared by disable FIFO break point)

Register Name	Bit Fields:	Description
CP_ENABLE Register 836	0:	Enable FIFO reads, reset value is "0" disable
	1:	FIFO break point enable bit, reset value is "0" disable
	2:	FIFO overflow interrupt enable, reset value is "0" disable
	3:	FIFO underflow interrupt enable, reset value is "0" disable
	4:	FIFO write pointer increment enable, reset value is "1" enable
	5:	FIFO break point interrupt enable, reset value is "0" disable
CP_CLEAR Register 838	0:	clear FIFO overflow interrupt
	1:	clear FIFO underflow interrupt
CP_STM_LOW Register 840	7:0	bits 7:0 of the Streaming Buffer low water mark in 32 bytes increment, default (reset) value is "0x0000"
CP_FIFO_BASEL 822	15:5	bits 15:5 of the FIFO base address in memory
CP_FIFO_BASE 822	9:0	bits 25:16 of the FIFO base address in memory
CP_FIFO_TOPL 824	15:5	bits 15:5 of the FIFO top address in memory
CP_FIFO_TOPH 824	9:0	bits 25:16 of the FIFO top address in memory
CP_FIFO_HICNTL 826	15:5	bits 15:5 of the FIFO high water count
CP_FIFO_HICNTH 826	9:0	bits 25:16 of the FIFO high water count
CP_FIFO_LOCNTL 828	15:5	bits 15:5 of the FIFO low water count
CP_FIFO_LOCNTH 828	9:0	bits 25:16 of the FIFO low water count
CP_FIFO_COUNTL 830	15:5	bits 15:5 of the FIFO_COUNT (entries currently in FIFO)
CP_FIFO_COUNTH 830	9:0	bits 25:16 of the FIFO_COUNT (entries currently in FIFO)
CP_FIFO_WPTRL 808	15:5	bits 15:5 of the FIFO write pointer
CP_FIFO_WPTRH 808	9:0	bits 25:15 of the FIFO write pointer
CP_FIFO_RPTRL 804	15:5	bits 15:5 of the FIFO read pointer
CP_FIFO_RPTRH 804	9:0	bits 25:15 of the FIFO read pointer
CP_FIFO_BRKL 832	15:5	bits 15:5 of the FIFO read address break point
CP_FIFO_BRKH 832	9:0	bits 9:0 if the FIFO read address break point

Other Example Compatible Implementations

Certain of the above-described system components 50 could be implemented as other than the home video game console configuration described above. For example, one could run graphics application or other software written for system 50

on a platform with a different configuration that emulates system 50 or is otherwise compatible with it. If the other platform can successfully emulate, simulate and/or provide some or all of the hardware and software resources of system 50, then the other platform will be able to successfully execute the software.

As one example, an emulator may provide a hardware and/or software configuration (platform) that is different from the hardware and/or software configuration (platform) of system 50. The emulator system might include software and/or hardware components that emulate or simulate some or all of hardware and/or software components of the system for which the application software was written. For example, the emulator system could comprise a general purpose digital computer such as a personal computer, which executes a software emulator program that simulates the hardware and/or firmware of system 50.

Some general purpose digital computers (e.g., IBM or MacIntosh personal computers and compatibles) are now equipped with 3D graphics cards that provide 3D graphics pipelines compliant with OpenGL, DirectX or other standard 3D graphics command APIs. They may also be equipped with stereophonic sound cards that provide high quality stereophonic sound based on a standard set of sound commands. Such multimedia-hardware-equipped personal computers running emulator software may have sufficient performance to approximate the graphics and sound performance of system 50. Emulator software controls the hardware resources on the personal computer platform to simulate the processing, 3D graphics, sound, peripheral and other capabilities of the home video game console platform for which the game programmer wrote the game software.

Figure 12A illustrates an example overall emulation process using a host platform 1201, an emulator component 1303, and a game software executable

binary image provided on a storage medium 62. Host 1201 may be a general or special purpose digital computing device such as, for example, a personal computer, a video game console, or any other platform with sufficient computing power. Emulator 1303 may be software and/or hardware that runs on host platform 1201, and provides a real-time conversion of commands, data and other information from storage medium 62 into a form that can be processed by host 1201. For example, emulator 1303 fetches "source" binary-image program instructions intended for execution by system 50 from storage medium 62 and converts these program instructions to a target format that can be executed or otherwise processed by host 1201.

As one example, in the case where the software is written for execution on a platform using an IBM PowerPC or other specific processor and the host 1201 is a personal computer using a different (e.g., Intel) processor, emulator 1303 fetches one or a sequence of binary-image program instructions from storage medium 1305 and converts these program instructions to one or more equivalent Intel binary-image program instructions. The emulator 1303 also fetches and/or generates graphics commands and audio commands intended for processing by the graphics and audio processor 114, and converts these commands into a format or formats that can be processed by hardware and/or software graphics and audio processing resources available on host 1201. As one example, emulator 1303 may convert these commands into commands that can be processed by specific graphics and/or or sound hardware of the host 1201 (e.g., using standard DirectX, OpenGL and/or sound APIs).

Certain emulators of system 50 might simply "stub" (i.e., ignore) some or all of the buffering and flow control techniques described above since they might have

much more memory resources than the example hardware implementation described above. Such emulators will typically respond to requests for buffer allocation by allocating memory resources, but might provide different flow control processing. Status and flow control requests as described above could be emulated by maintaining an emulated state of the hardware, and using that state to respond to the status requests.

An emulator 1303 used to provide some or all of the features of the video game system described above may also be provided with a graphic user interface (GUI) that simplifies or automates the selection of various options and screen modes for games run using the emulator. In one example, such an emulator 1303 may further include enhanced functionality as compared with the host platform for which the software was originally intended.

Figure 12B illustrates an emulation host system 1201 suitable for use with emulator 1303. System 1201 includes a processing unit 1203 and a system memory 1205. A system bus 1207 couples various system components including system memory 1205 to processing unit 1203. System bus 1207 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System memory 1207 includes read only memory (ROM) 1252 and random access memory (RAM) 1254. A basic input/output system (BIOS) 1256, containing the basic routines that help to transfer information between elements within personal computer system 1201, such as during start-up, is stored in the ROM 1252. System 1201 further includes various drives and associated computer-readable media. A hard disk drive 1209 reads from and writes to a (typically fixed) magnetic hard disk 1211. An additional (possible optional) magnetic disk drive 1213 reads from and writes to a

removable "floppy" or other magnetic disk 1215. An optical disk drive 1217 reads from and, in some configurations, writes to a removable optical disk 1219 such as a CD ROM or other optical media. Hard disk drive 1209 and optical disk drive 1217 are connected to system bus 1207 by a hard disk drive interface 1221 and an optical drive interface 1225, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules, game programs and other data for personal computer system 1201. In other configurations, other types of computer-readable media that can store data that is accessible by a computer (e.g., magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like) may also be used.

A number of program modules including emulator 1303 may be stored on the hard disk 1211, removable magnetic disk 1215, optical disk 1219 and/or the ROM 1252 and/or the RAM 1254 of system memory 1205. Such program modules may include an operating system providing graphics and sound APIs, one or more application programs, other program modules, program data and game data. A user may enter commands and information into personal computer system 1201 through input devices such as a keyboard 1227, pointing device 1229, microphones, joysticks, game controllers, satellite dishes, scanners, or the like. These and other input devices can be connected to processing unit 1203 through a serial port interface 1231 that is coupled to system bus 1207, but may be connected by other interfaces, such as a parallel port, game port Fire wire bus or a universal serial bus (USB). A monitor 1233 or other type of display device is also connected to system bus 1207 via an interface, such as a video adapter 1235.

System 1201 may also include a modem 1154 or other network interface means for establishing communications over a network 1152 such as the Internet. Modem 1154, which may be internal or external, is connected to system bus 123 via serial port interface 1231. A network interface 1156 may also be provided for allowing system 1201 to communicate with a remote computing device 1150 (e.g., another system 1201) via a local area network 1158 (or such communication may be via wide area network 1152 or other communications path such as dial-up or other communications means). System 1201 will typically include other peripheral output devices, such as printers and other standard peripheral devices.

In one example, video adapter 1235 may include a 3D graphics pipeline chip set providing fast 3D graphics rendering in response to 3D graphics commands issued based on a standard 3D graphics application programmer interface such as Microsoft's DirectX 7.0 or other version. A set of stereo loudspeakers 1237 is also connected to system bus 1207 via a sound generating interface such as a conventional "sound card" providing hardware and embedded software support for generating high quality stereophonic sound based on sound commands provided by bus 1207. These hardware capabilities allow system 1201 to provide sufficient graphics and sound speed performance to play software stored in storage medium 62.

All documents referenced above are hereby incorporated by reference.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the scope of the appended claims.

We Claim:

1. A graphics system including:

a producer that outputs graphics commands,

a consumer that consumes the graphics commands outputted by the producer, and

a storage device coupled between the producer and the consumer, the storage device storing at least one buffer that receives and temporarily stores graphics commands outputted by the producer for delivery to the consumer, wherein the producer and the consumer are capable of accessing said buffer independently of one another.

2. A graphics system as in claim 1 wherein the producer and the consumer have independent read and/or write pointers.

3. A graphics system as in claim 1 wherein the storage device stores plural variable sized buffers disposed at selected locations within the storage device.

4. A graphics system as in claim 1 wherein the storage device stores plural buffers each of which can be independently accessed by the producer and/or the consumer.

5. The graphics system of claim 4 wherein the consumer is incapable of writing to at least an active one of the plural buffers, but maintains, independently of the producer, a write pointer for at least said active one of the plural buffers.

6. The graphics system of claim 2 wherein the consumer selectively increments the consumer write pointer in response to the producer writing to the active buffer.

7. The graphics system of claim 4 wherein the producer provides a producer read pointer and a producer write pointer associated with a first of said plural buffers, and the consumer independently maintains a consumer read pointer and a consumer write pointer associated with said first of said plural buffers.

8. The graphics system of claim 6 wherein the consumer increments the consumer read pointer as the consumer reads from an active buffer, and suspends reading from the active buffer when the incremented consumer read pointer has a predetermined relationship with the consumer write pointer.

9. The graphics system of claim 4 wherein a first of the plural buffers includes a read command controlling the consumer to consume a set of graphics commands the producer stores elsewhere within the storage device, and to resume consuming graphics commands from the first buffer after consuming the graphics commands stored elsewhere.

10. The graphics system of claim 9 wherein the read command specifies a starting address and a length of a display list, the read command controlling the consumer to read the display list of the specified length beginning at the specified starting address.

11. The graphics system of claim 1 wherein the buffer provides circular first-in-first-out access.

12. The graphics system of claim 1 wherein the buffer provides linear first-in-first-out access.

13. The graphics system of claim 1 wherein the buffer can be selectively attached to both the producer and the consumer simultaneously.

14. The graphics system of claim 1 wherein the buffer is attached to the producer while a further buffer is attached to the consumer.

15. The system of claim 1 wherein the buffer is first attached to the producer, and is subsequently detached from the producer and attached to the consumer.

16. The system of claim 1 wherein the buffer can be attached to the producer, the consumer, or both.

17. The system of claim 1 wherein only one buffer is attached to the producer at a time.

18. The system of claim 1 wherein only one buffer is attached to the consumer at a time.

19. The system of claim 1 wherein the buffer has a maximum size of 16 KB.

20. The system of claim 1 wherein the producer sets the size of the buffer.

21. The system of claim 1 wherein the buffer is dynamically sized to store a frame of graphics commands.

22. The system of claim 1 wherein the producer declares the buffer by issuing a graphics buffer initialization command specifying buffer starting address and buffer length.

23. The system of claim 1 wherein the buffer has a length that is a multiple of 32 bytes and has a minimum size of 64 KB.

24. The graphics system of claim 1 wherein the producer may write a breakpoint into the buffer, the consumer suspending consumption of graphics commands upon encountering the breakpoint.

25. The system of claim 1 wherein the buffer has an overflow status indicator indicating when the producer overwrites a location therein.

26. The system of claim 1 further including a hardware status register that indicates the status of the buffer.

27. The system of claim 26 wherein the status register includes the following parameters:

position of a producer write pointer relative to buffer full and buffer empty;

buffer overflow;

whether the producer is currently writing into the buffer; and

whether the consumer is currently reading from the buffer.

28. The system of claim 1 further including a hardware controller coupled to the buffer, the hardware controller providing flow control logic to prevent writes from overrunning reads.

29. The system of claim 1 further including a hardware controller coupled to the buffer, the hardware controller wrapping read and write pointers from a last location to a first location thereof.

30. The graphics system of claim 1 wherein the producer comprises a processor and the consumer comprises a graphics processor including a graphics pipeline.

31. The system of claim 1 wherein the storage device comprises a main system memory, and the producer dynamically allocates said buffer within the main system memory.

32. A graphics system including:

a storage buffer that receives and temporarily stores graphics commands;

a producer that writes graphics commands into said buffer, said producer maintaining a producer write pointer and a producer read pointer associated with the buffer; and

a consumer that consumes the graphics commands stored within the buffer, the consumer maintaining a consumer write pointer that is independent of the producer write pointer, and a consumer read pointer that is independent of the producer read pointer.

33. The graphics system of claim 32 wherein the consumer increments the consumer read pointer each time the consumer consumes from the buffer, and suspends consumption from the buffer when the consumer read pointer has a predetermined relationship with the consumer write pointer.

34. The graphics system of claim 32 wherein the consumer selectively auto-increments the consumer write pointer in response to the producer writing to the buffer.

35. The graphics system of claim 32 wherein the producer sends the consumer a configuration command specifying whether the consumer should auto-increment the consumer write pointer in response to producer writes to the buffer.

36. In a graphics system including a graphics command producer that writes graphics commands into a buffer based on a producer write pointer and a graphics command consumer that reads graphics commands from the buffer based on a consumer read pointer,

an improvement comprising:

a consumer write pointer independently maintained by the consumer, the consumer write pointer indicating the extent of valid data the producer has written into said buffer, the consumer ceasing to consume graphics commands from the buffer upon the consumer read pointer having a predetermined relationship to the consumer write pointer.

37. In an interactive graphics system including a processor module executing an application, a graphics processor module and at least one memory coupled to the processor module and to the graphics processor module, a method of controlling the flow of graphics commands between the processor module and the graphics processor module comprising:

dynamically establishing, under control of the application, a variable number of FIFO buffers in the memory, the application specifying the size of each of the FIFO buffers;

the application controlling the processor module to write graphics commands into at least a first of the plurality of FIFO buffers; and

the application sending graphics commands to the graphics processor module that control the graphics processor module to read independently of said processor writes, the graphics command from the first FIFO buffer.

38. The graphics system of claim 37 wherein the processor module provides a processor module read pointer and a processor module write pointer associated with a first of said plurality of buffers, and the graphics processor module independently maintains a graphics processor module read pointer and a graphics processor module write pointer associated with said first buffer.

39. The graphics system of claim 37 wherein the graphics processor module increments the graphics processor module read pointer each time the graphics processor module reads from the first buffer, and suspends reading from the first buffer when the graphics processor module read pointer has a predetermined relationship with the graphics processor module write pointer.

40. The graphics system of claim 37 wherein the graphics processor module selectively auto-increments the graphics processor module write pointer in response to the processor module writing to the first buffer.

41. The graphics system of claim 37 wherein the graphics processor module maintains, independently of the processor module, a write pointer for at least an active one of the plurality of buffers.

42. The method of claim 37 further including setting a breakpoint, and at least temporarily suspending the graphics processor module from reading a buffer in response to the graphics processor module encountering the breakpoint.

43. The method of claim 37 wherein the selectively controlling step includes suspending the processor module from writing to the buffer upon detection of an overflow.

44. A method of controlling the flow of graphics data comprising:

writing graphics data into plural variable sized FIFO buffers each having plural storage locations;

setting a breakpoint associated with at least one of the plural storage locations;

reading graphics data from the plural buffers in a predetermined order;

temporarily suspending the reading step upon encountering the at least one location associated with the breakpoint, and generating an interrupt; and

resuming the reading step in response to receipt of a clear interrupt command.

45. The method of claim 44 wherein the resuming step is performed in response to receipt of a disable breakpoint command.

46. The method of claim 44 wherein:

the writing step comprises writing graphics data corresponding to a first image frame into a first part of the buffer, and writing graphics data corresponding to a second image frame into a second part of the buffer; and

the setting step comprises setting the breakpoint to be associated with a location separating the first and second buffer portions.

47. The method of claim 44 wherein the predetermined order is first-in-first-out.

48. The method of claim 44 wherein the writing step is performed by general purpose processing hardware, and the reading step is performed by special purpose graphics hardware.

49. The method of claim 44 wherein the reading and writing steps are performed substantially simultaneously.

50. The method of claim 44 wherein the writing step completes before the reading step begins.

51. The method of claim 44 wherein the resumed reading step continues until a further breakpoint is encountered, the buffer is empty, or an image frame represented by the graphics data is aborted.

52. The method of claim 44 further including indicating the status of the buffer.

53. The method of claim 52 wherein the status indicating step includes indicating at least three of the following parameters:

position of a write pointer relative to buffer full and buffer empty;

buffer overflow;

the writing step is active;

whether command processing is idle; and

the reading step is active.

54. The method of claim 44 wherein the writing step comprises writing the buffer circularly, and the reading step comprises reading the buffer circularly.

55. A graphics system including:

a storage device that receives and temporarily stores graphics commands;

a producer that writes commands into a buffer within said storage device, said commands including a first set of graphics commands and a command referencing a second set of graphics commands stored elsewhere within said storage device; and

a consumer that consumes the first set of graphics commands stored within the buffer and, in response to encountering the referencing command, consumes the second set of graphics commands referenced thereby and subsequently returns to the buffer to consume additional commands therefrom.

56. The graphics system of claim 55 wherein the buffer is a circular buffer.

57. The graphics system of claim 55 wherein the referencing command specifies a starting address of a display list, the referencing command controlling the consumer to read the display list beginning at the specified starting address.

58. The graphics system of claim 55 wherein the referencing command specifies a number of data units the consumer is to consume.

59. The graphics system of claim 55 wherein the consumer is incapable of writing to the buffer, but maintains, independently of the producer, a write pointer for the buffer.

60. In a graphics system, a method for passing graphics commands from a producer of graphics commands to a consumer of graphics commands, the method comprising:

creating plural variable sized buffers disposed at variable locations within a memory coupled to the producer and the consumer,

temporarily storing graphics commands produced by the producer in the variable sized buffers;

consuming the graphics commands from the variable sized buffers with the consumer by accessing the buffers independently of the producer; and

generating at least a part of a graphics image based at least in part on the consumed graphics commands.

61. The method of claim 60 wherein the consumer is incapable of writing to at least an active one of the plural buffers, and the method further includes the consumer maintaining, independently of the producer, a write pointer for at least said active one of the plural buffers.

62. The method of claim 60 further including maintaining, with the producer, a producer read pointer and a producer write pointer associated with a first of said plural buffers, and the independently maintaining, with the consumer, a consumer read pointer and a consumer write pointer associated with said first of said plural buffers.

63. The graphics system of claim 62 further including incrementing, with the consumer, a consumer read pointer as the consumer reads from an active buffer, and suspending reading from the active buffer when the incremented consumer read pointer has a predetermined relationship with the consumer write pointer.

64. The graphics system of claim 63 further including selectively incrementing the consumer write pointer in response to the producer writing to the active buffer.

65. The method of claim 60 wherein a first of the plural buffers includes a read command, and the method further includes:

(a) consuming a set of graphics commands the producer stores elsewhere within the storage device in response to encountering the read command, and

(b) resuming consumption of graphics commands from the first buffer after consuming the graphics commands stored elsewhere.

66. The graphics system of claim 65 wherein the read command specifies a starting address and a length of a display list, and step (a) includes controlling the consumer to read the display list of the specified length beginning at the specified starting address.

67. The method of claim 60 wherein a first of the plural buffers provides circular first-in-first-out access.

68. The method of claim 60 wherein a first of the plural buffers provides linear first-in-first-out access.

69. The method of claim 60 further including selectively attaching any of the plural buffers to both the producer and the consumer simultaneously.

70. The method of claim 60 further including attaching a first of the plural buffers to the producer and attaching a second of the plural buffers to the consumer.

71. The system of claim 60 further including attaching a first of the plural buffers to the producer, and subsequently detaching the first buffer from the producer and attaching the first buffer to the consumer.

72. The system of claim 60 further including attaching any of the plural buffers to the producer, the consumer, or both.

73. The system of claim 60 further including attaching only one of the plural buffers to the producer at a time.

74. The system of claim 60 further including attaching only one of the plural buffers to the consumer at a time.

75. A method for producing images including:

maintaining a producer write pointer and a producer read pointer associated with a buffer;

writing graphics commands into the buffer, and updating at least the write pointer in response to the writing;

maintaining, in association with the buffer, a consumer write pointer that is independent of the producer write pointer, and a consumer read pointer that is independent of the producer read pointer;

consuming the graphics commands stored within the buffer, and updating at least the read pointer in response to the consuming; and

producing at least a part of a graphics image at least in part in response to the consuming step.

76. A method of producing images including:

writing commands into a buffer within a storage device, said commands including a first set of graphics commands and a command referring to a second set of graphics commands stored elsewhere within said storage device;

consuming the first set of graphics commands stored within the buffer;

in response to encountering the referring command, consuming the second set of graphics commands and subsequently automatically returning to consume additional commands from the buffer; and

generating at least a part of an image at least in part in response to the consumed first and second sets of graphics commands.

77. The method of claim 76 wherein the second set of graphics commands comprises a display list.

78. A method of generating data structures comprising:

writing a graphics command stream into a command stream buffer, said command stream including a predetermined command signifying the beginning of a display list; and

in response to the predetermined command, redirecting the graphics command stream from the command stream buffer to a display list buffer.

79. The method of claim 78 wherein the command stream buffer comprises a first-in-first-out buffer.

80. The method of claim 78 further including writing a further predetermined command that redirects the graphics command stream from the display list buffer back to the command stream buffer.

81. A method of supplying 3D graphics commands to a 3D graphics command consumer comprising:

(a) storing a command sequence beginning at a predetermined storage location; and

(b) supplying a graphics command stream through a FIFO buffer to the consumer, the stream including at least one command that refers the command consumer to the predetermined storage location,

wherein the producer returns to the FIFO buffer after consuming the command sequence beginning at the predetermined storage location.

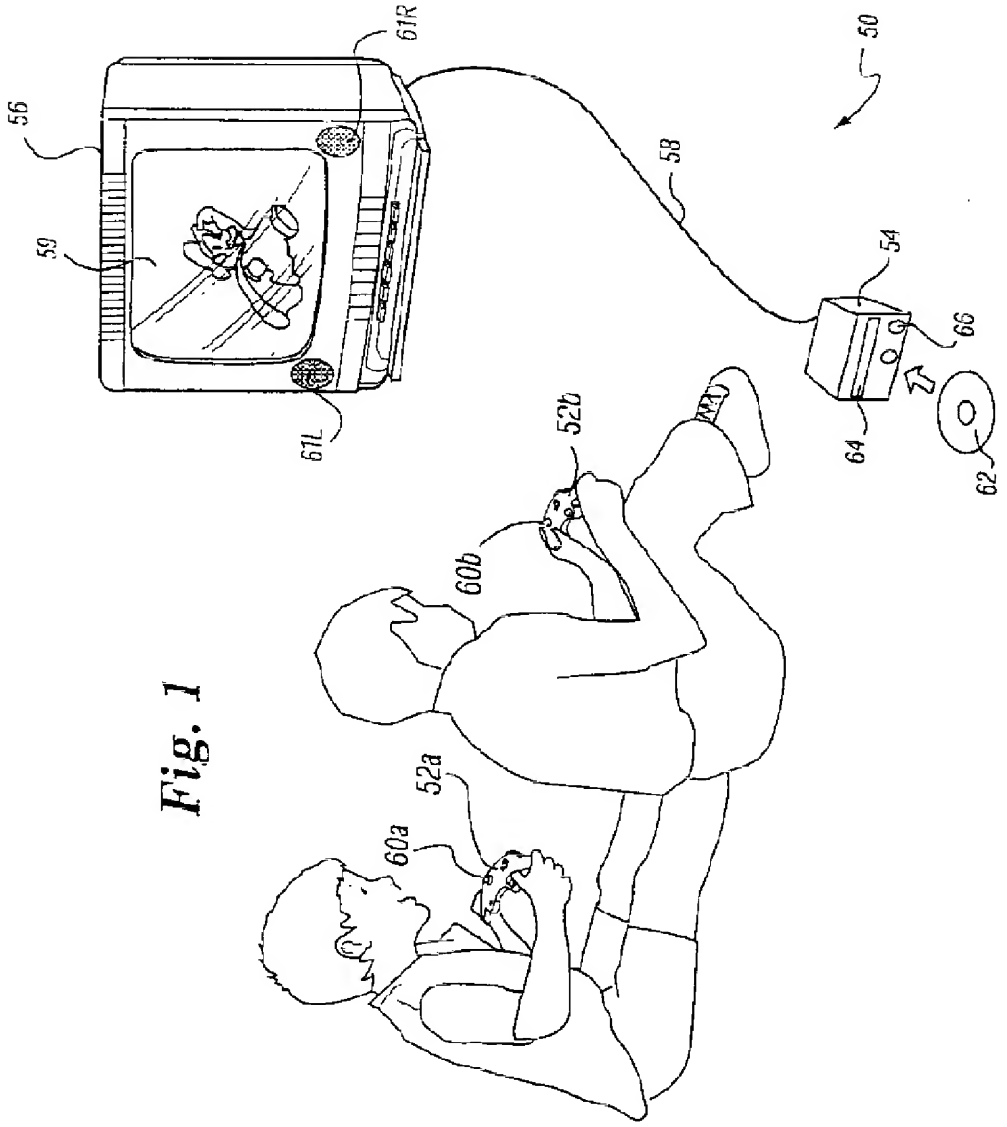


Fig. 1

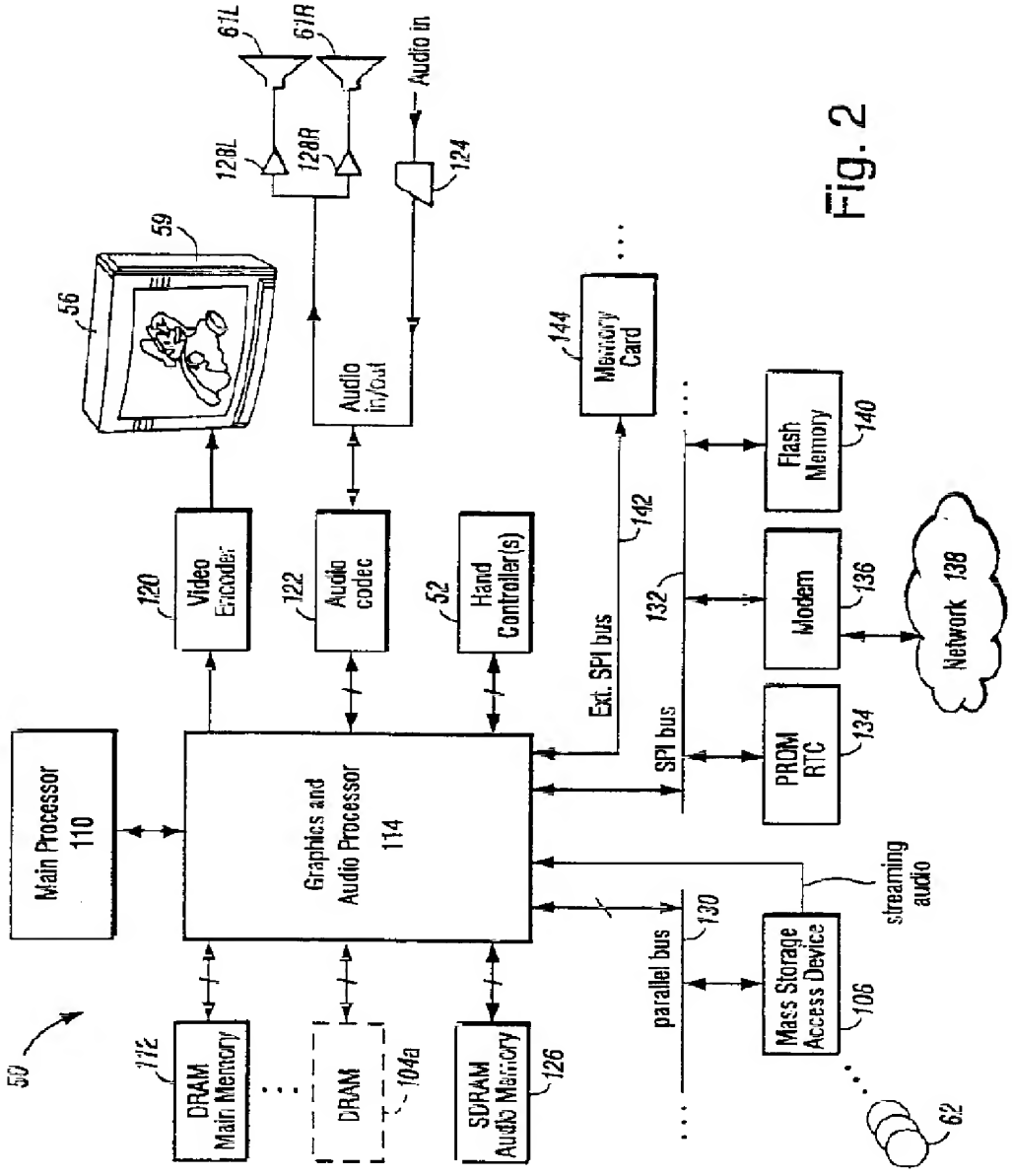

50
Main Processor 110
Graphics and Audio Processor 114
DRAM Main Memory 112
DRAM 104a
SDRAM Audio Memory 126
Video Encoder 120
Audio codec 122
Hand Controller(s) 52
parallel bus 130
Ext. SPI bus 142
SPI bus 132
Mass Storage Access Device 106
PROM RTC 134
Modem 136
Flash Memory 140
Memory Card 144
Network 138
streaming audio
62
56
59
Audio in/out
128L
128R
61L
61R
124
Audio in

Fig. 2

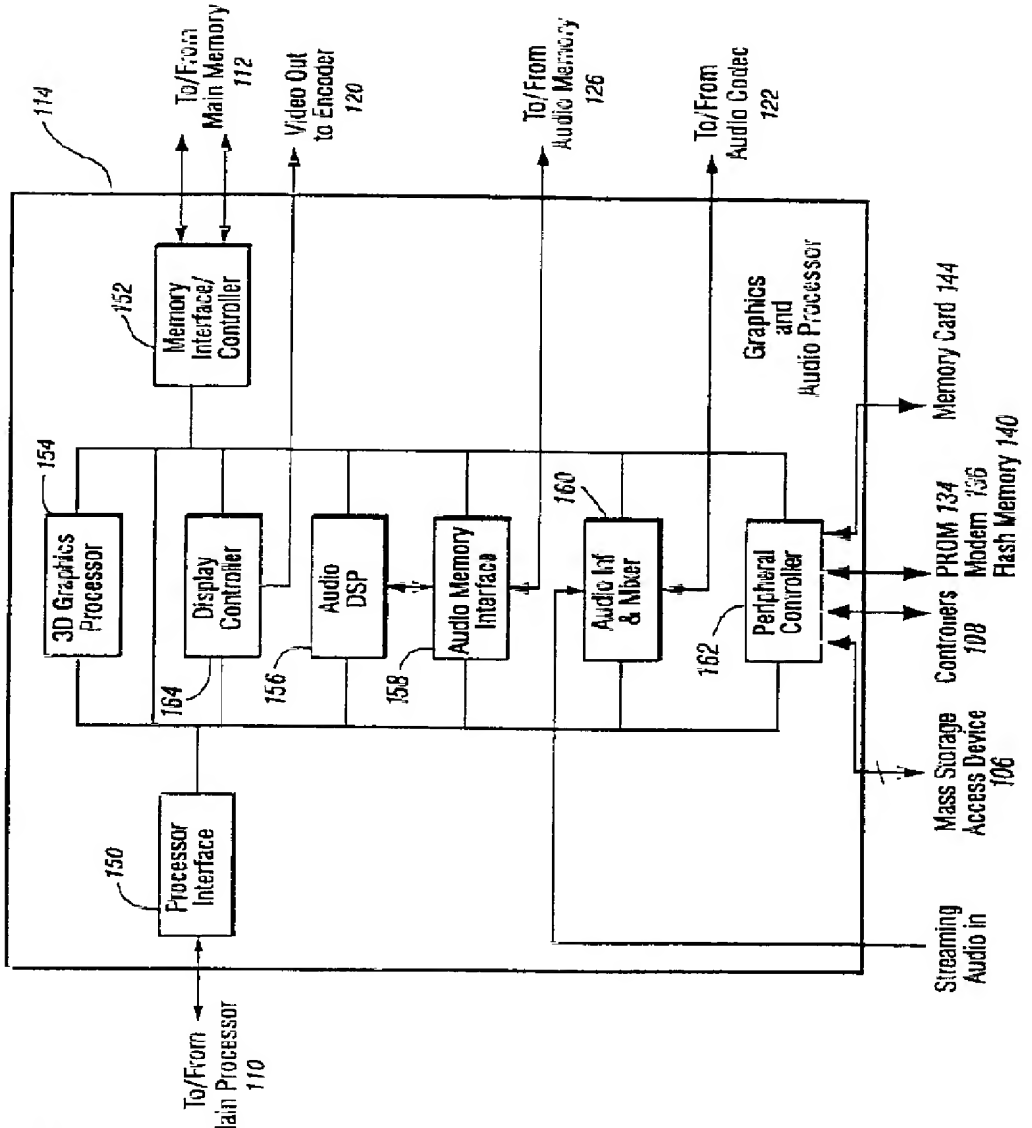

114
To/From Main Memory 112
Video Out to Encoder 120
To/From Audio Memory 126
To/From Audio Codec 122
Memory Interface/ Controller
152
Graphics and Audio Processor
Memory Card 144
154
3D Graphics Processor
Display Controller
Audio DSP
Audio Memory Interface
160
Audio Int & Mixer
Peripheral Controller
PROM 134
Modem 136
Flash Memory 140
164
156
158
162
Controllers 108
Mass Storage Access Device 106
Processor Interface
150
Streaming Audio in
To/From Main Processor 110

Fig. 3

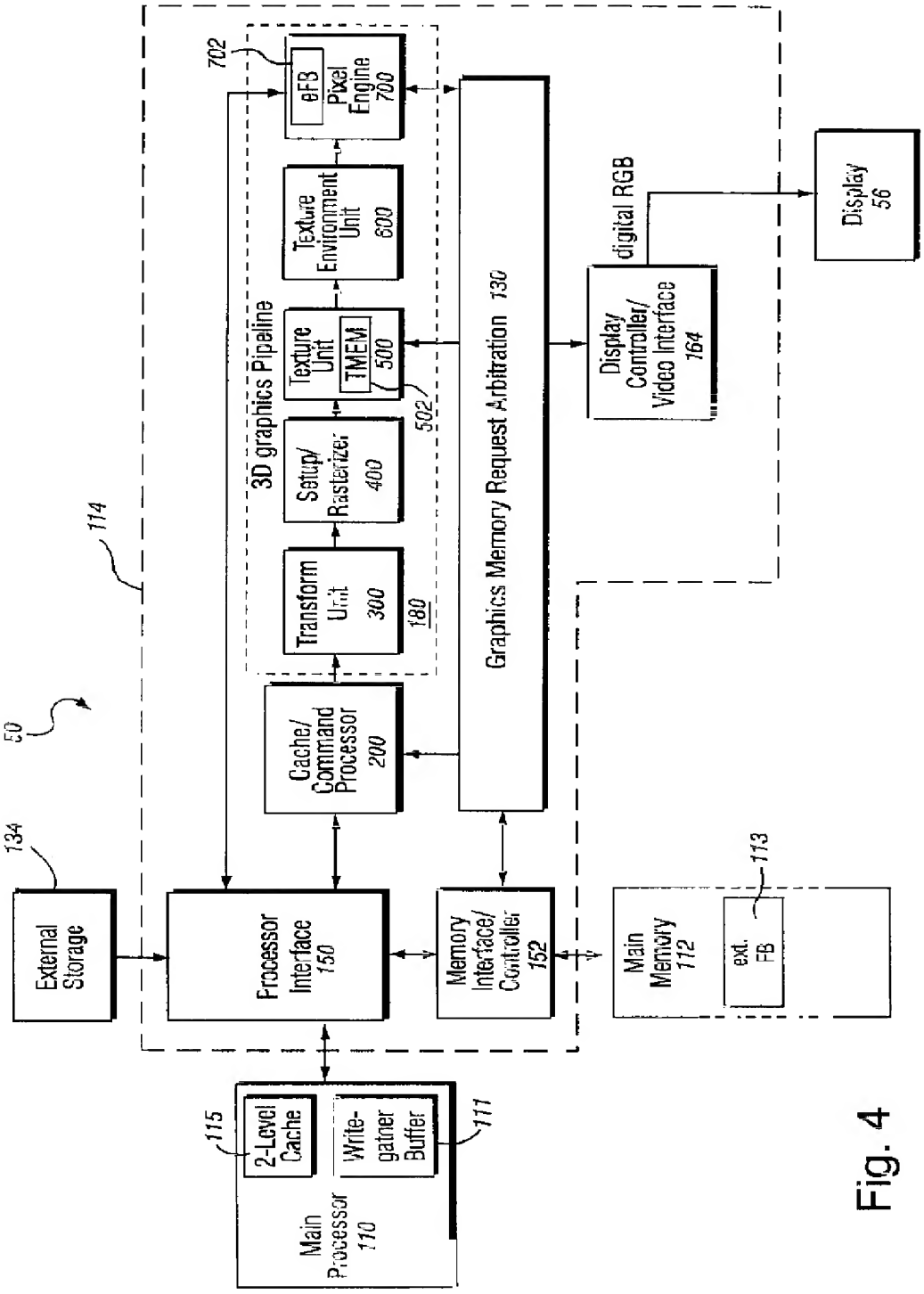

50
114
134
External Storage
Processor Interface 150
Cache/ Command Processor 200
3D graphics Pipeline
Transform Unit 300
180
Setup/ Rasterizer 400
Texture Unit
TMEM 500
502
Texture Environment Unit 600
702
eFB
Pixel Engine 700
Graphics Memory Request Arbitration 130
Display Controller/ Video Interface 164
digital RGB
Display 56
Memory Interface/ Controller 152
Main Memory 112
ext. FB
113
115
2-Level Cache
Write-gather Buffer
111
Main Processor 110

Fig. 4

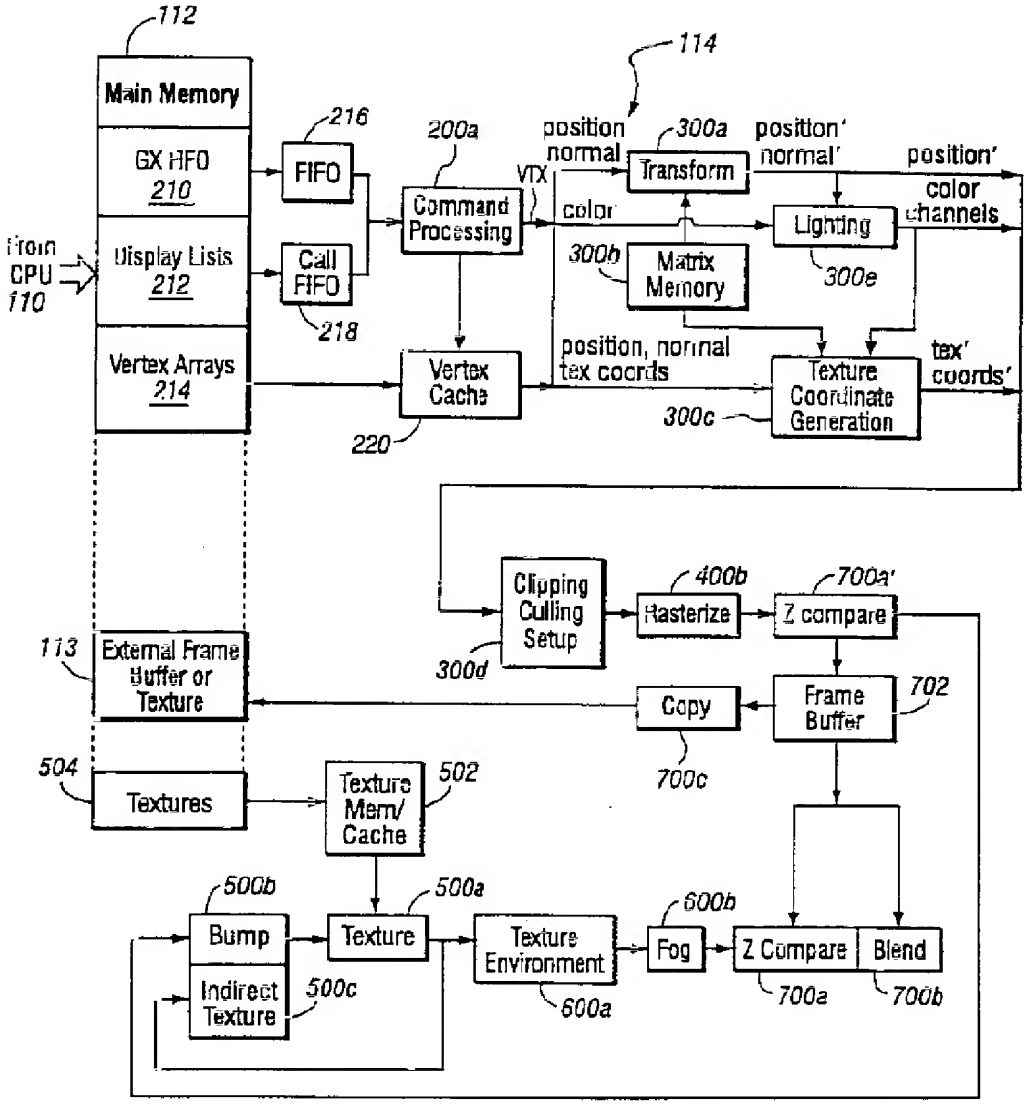


Fig. 5 EXAMPLE GRAPHICS PROCESSOR FLOW

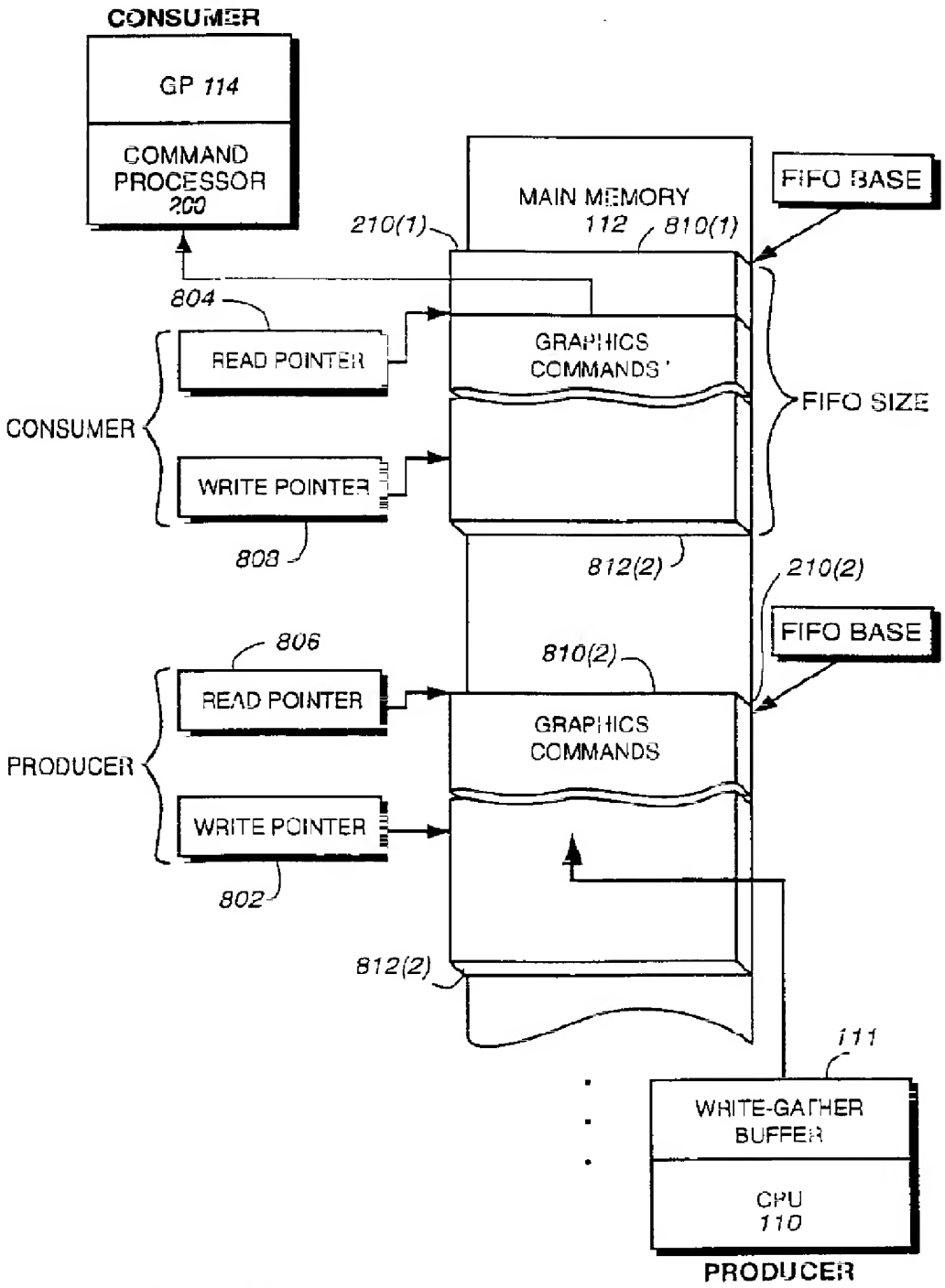


Fig. 6 EXAMPLE MULTI - BUFFERING

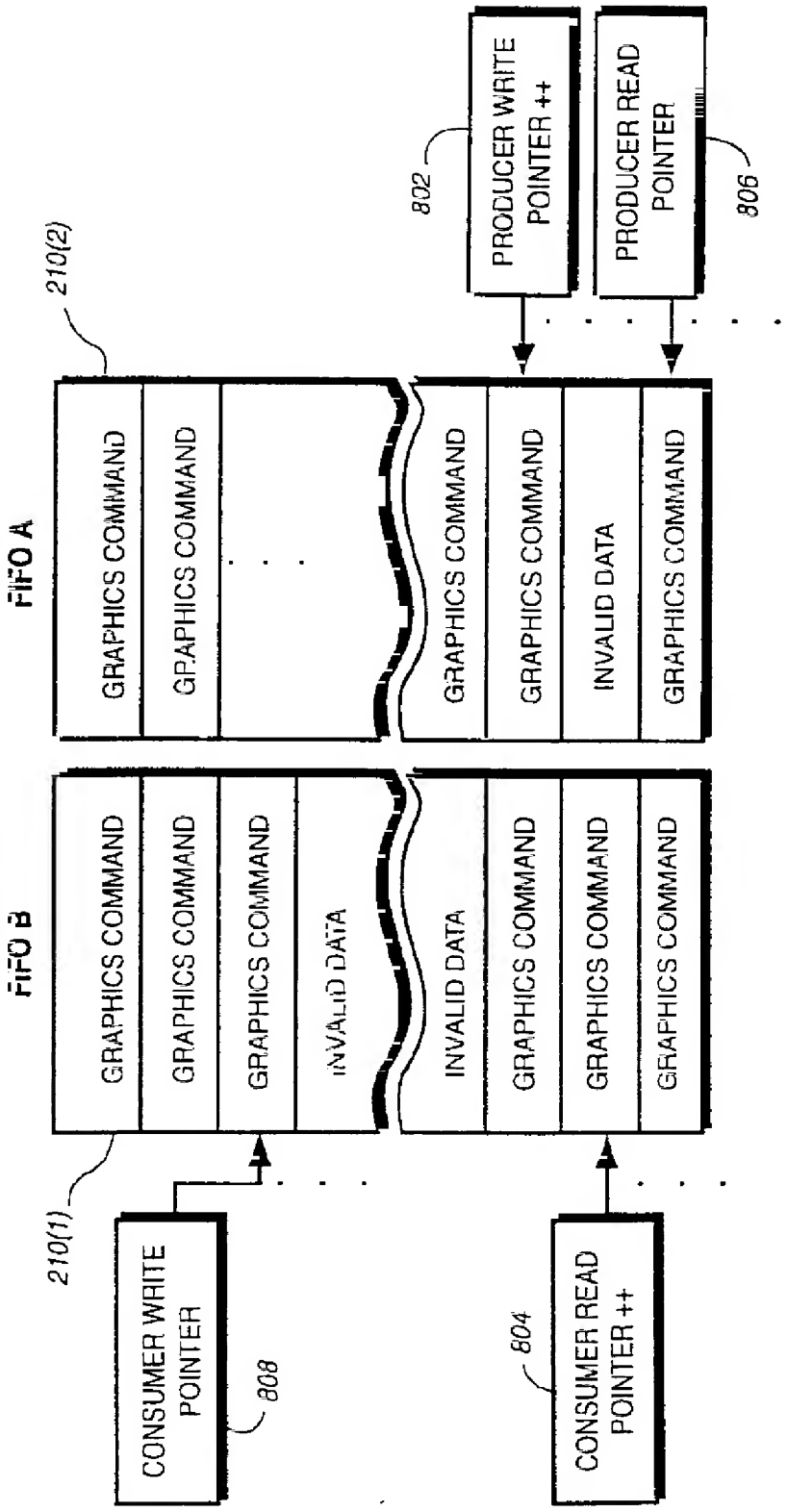


Fig. 7 EXAMPLE INDEPENDENT CONSUMER AND PRODUCER READ AND WRITE POINTERS

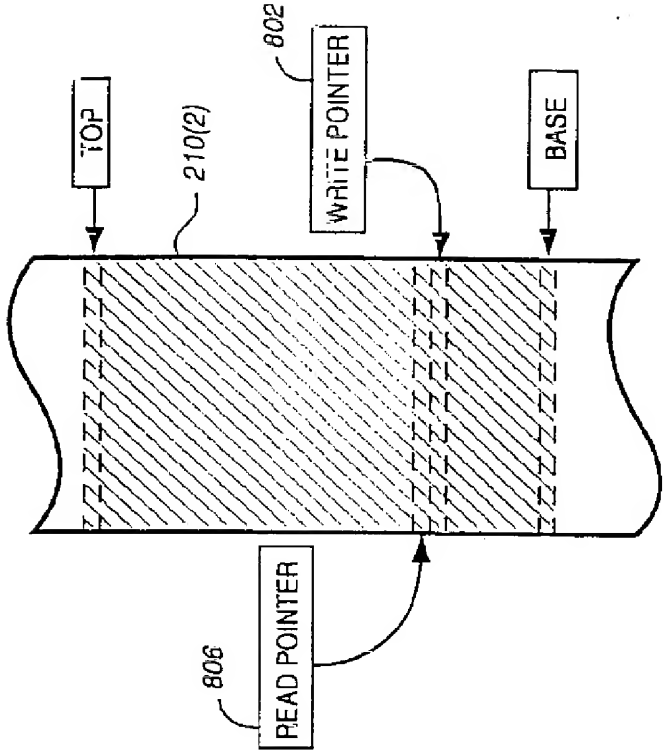


Fig. 8B FULL CONDITION

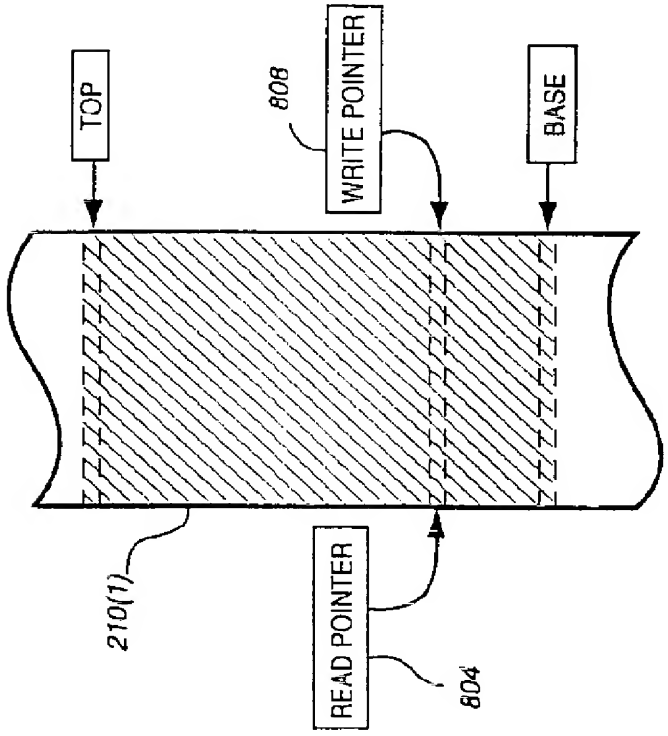


Fig. 8A EMPTY CONDITION

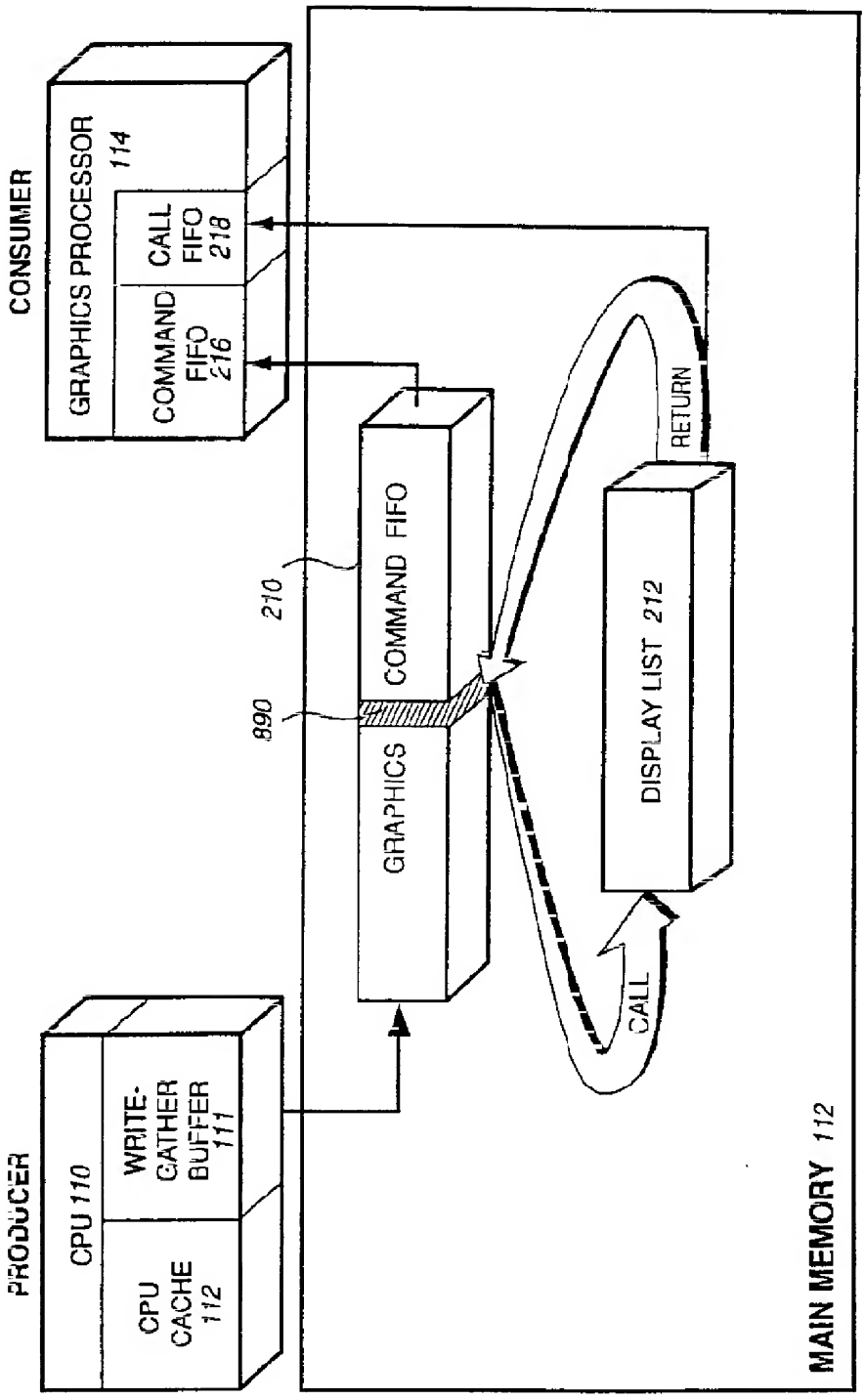


Fig. 9 EXAMPLE DISPLAY LIST CALL FROM COMMAND FIFO

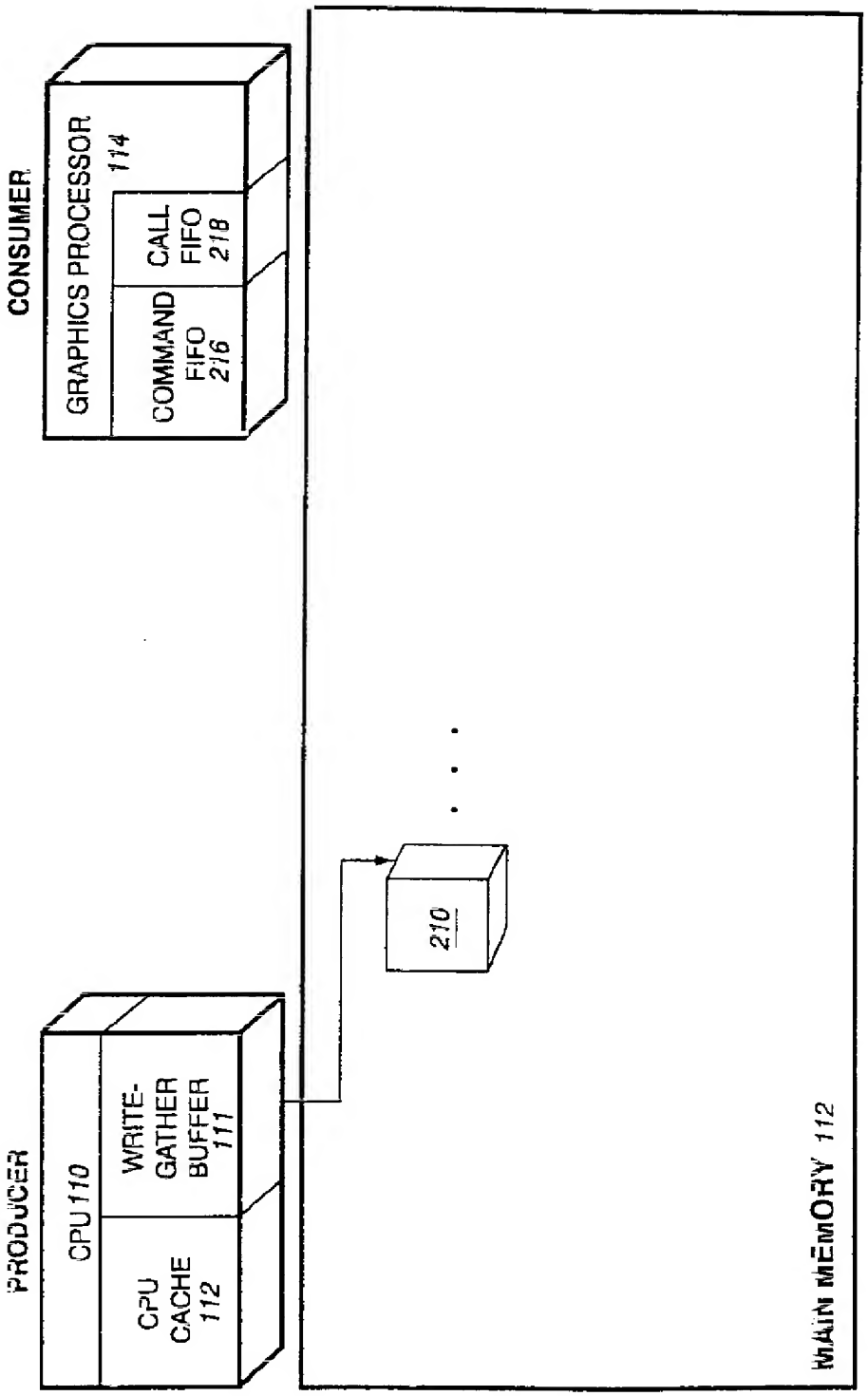


Fig. 10A EXAMPLE WRITE TO FIFO BUFFER

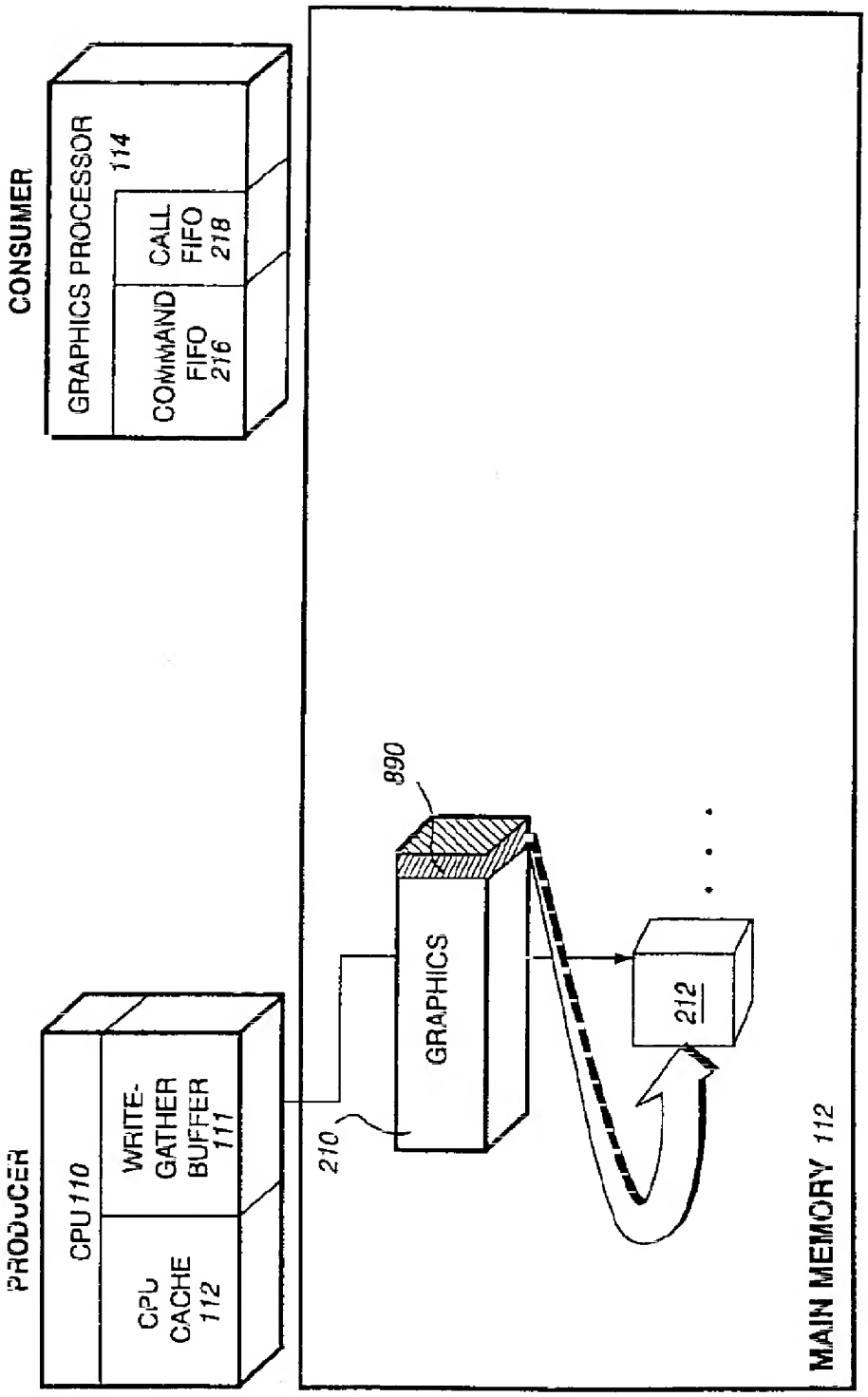


Fig. 10B EXAMPLE BEGIN DISPLAY LIST COMMAND

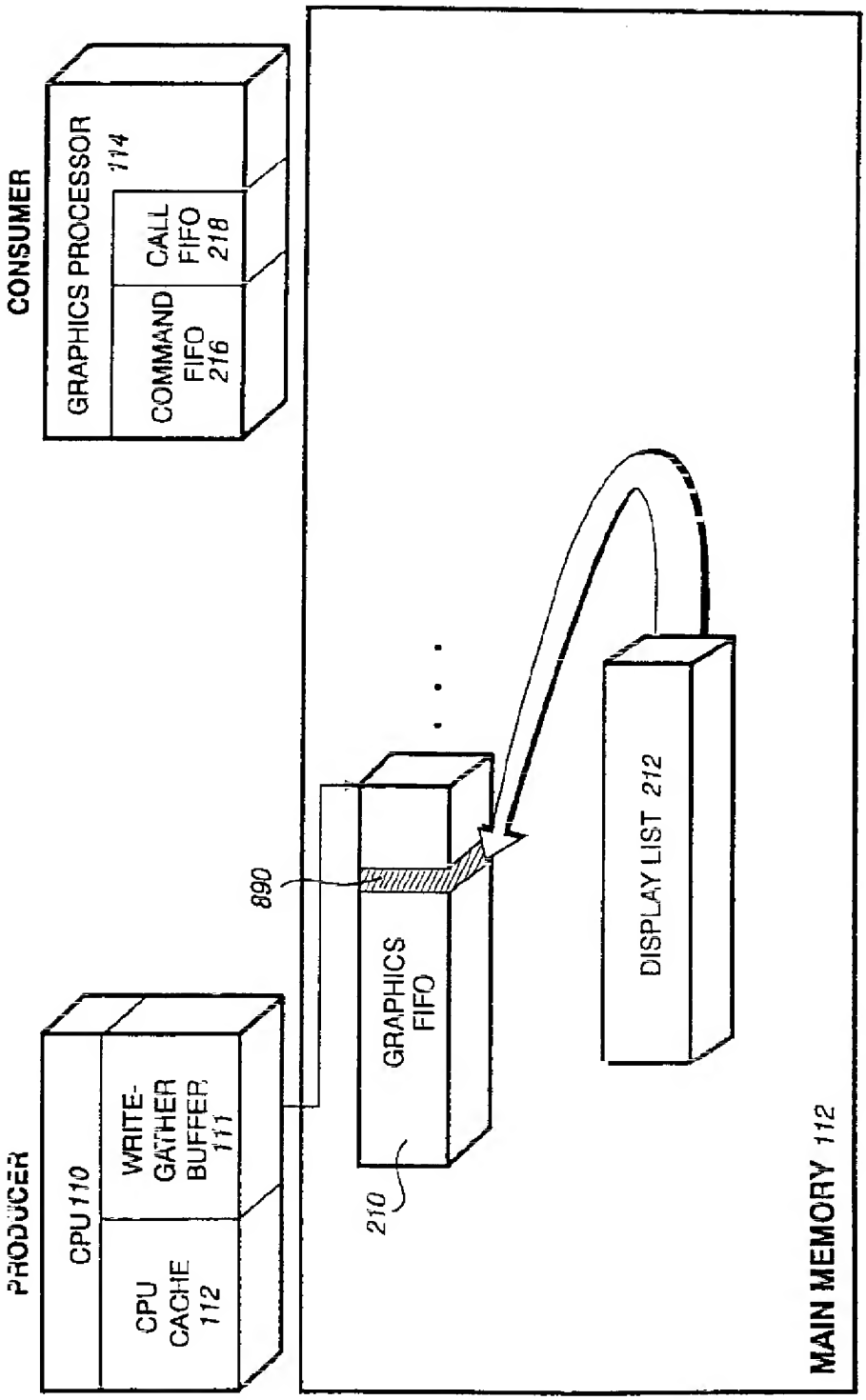


Fig. 10C EXAMPLE END DISPLAY LIST COMMAND

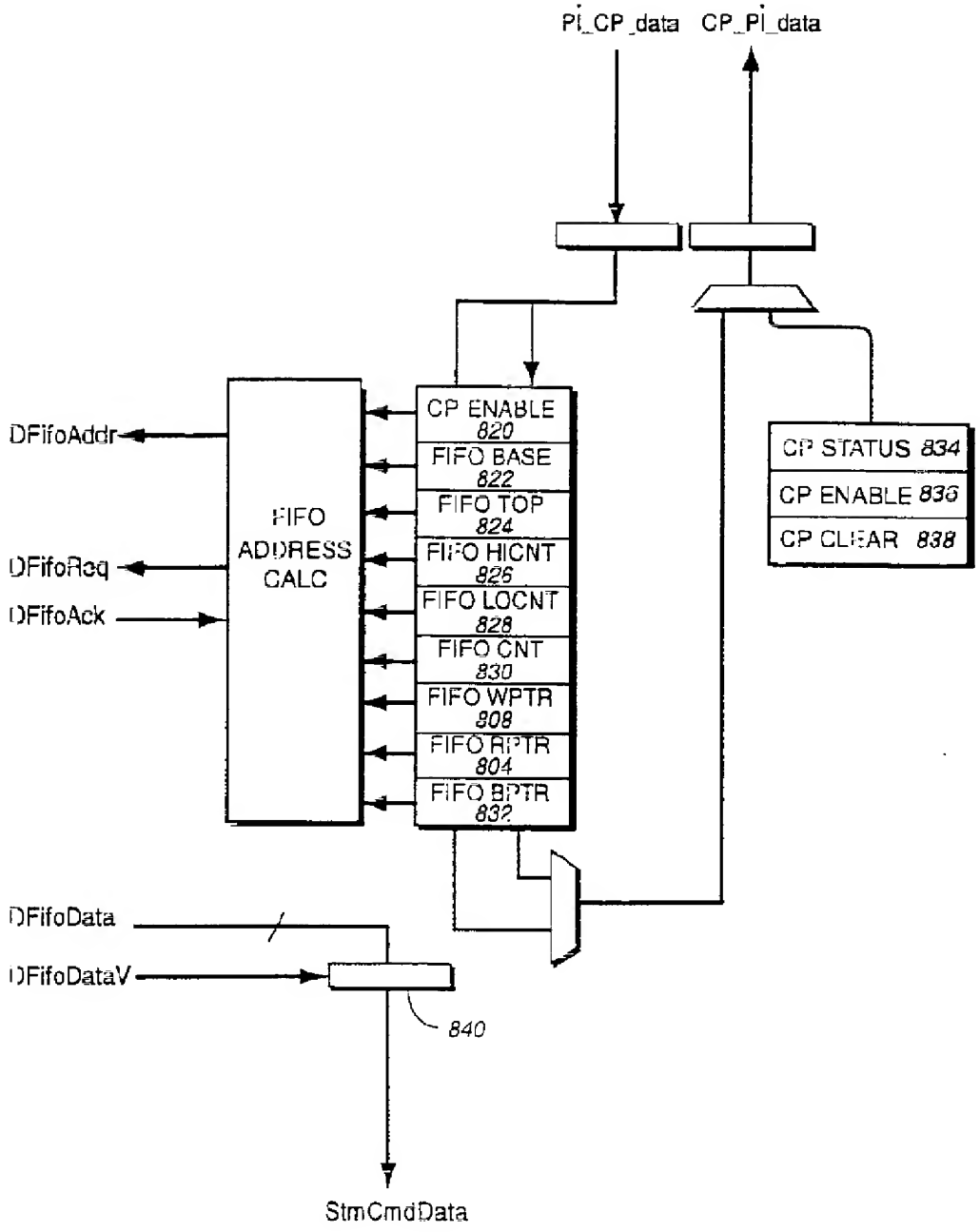


Fig. 11 EXAMPLE FIFO MANAGER

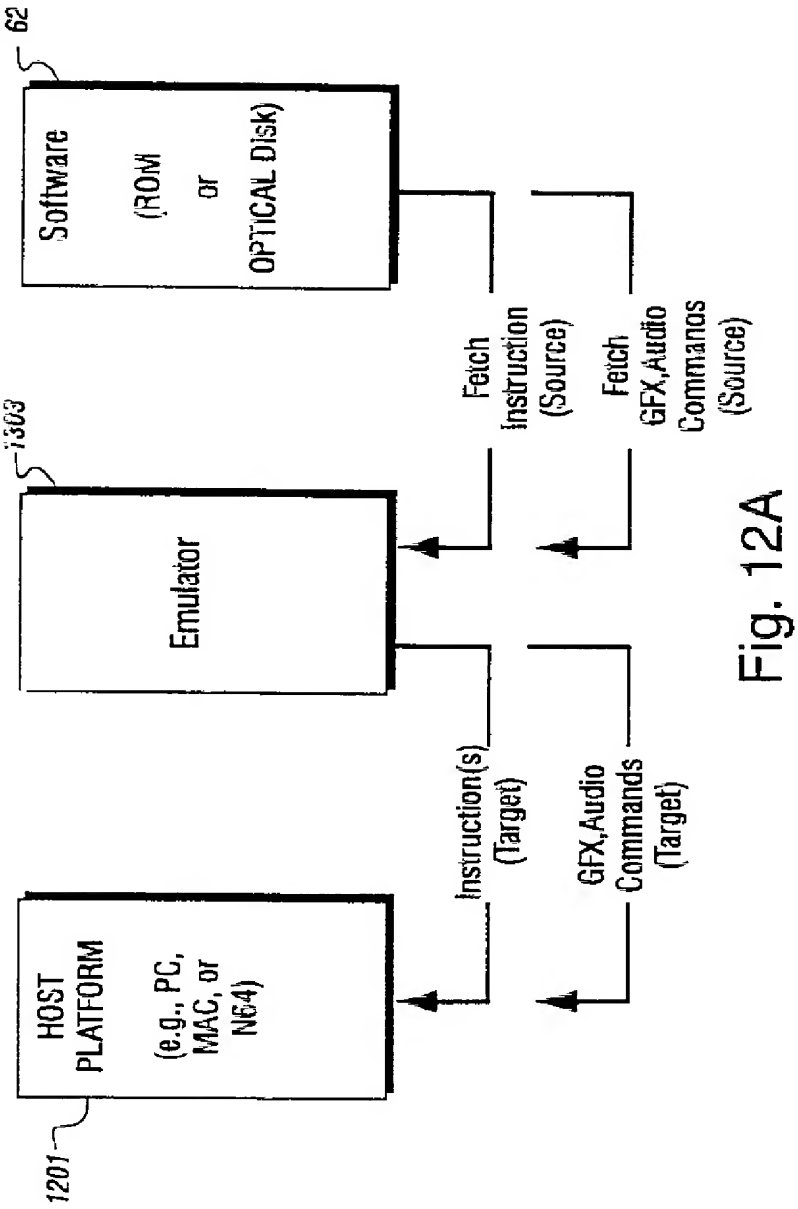

62
Software
(ROM
or
OPTICAL Disk)
1303
Emulator
Fetch
Instruction
(Source)
Fetch
GFX,Audio
Commands
(Source)
Instruction(s)
(Target)
GFX,Audio
Commands
(Target)
HOST
PLATFORM
(e.g., PC,
MAC, or
N64)
1201

Fig. 12A

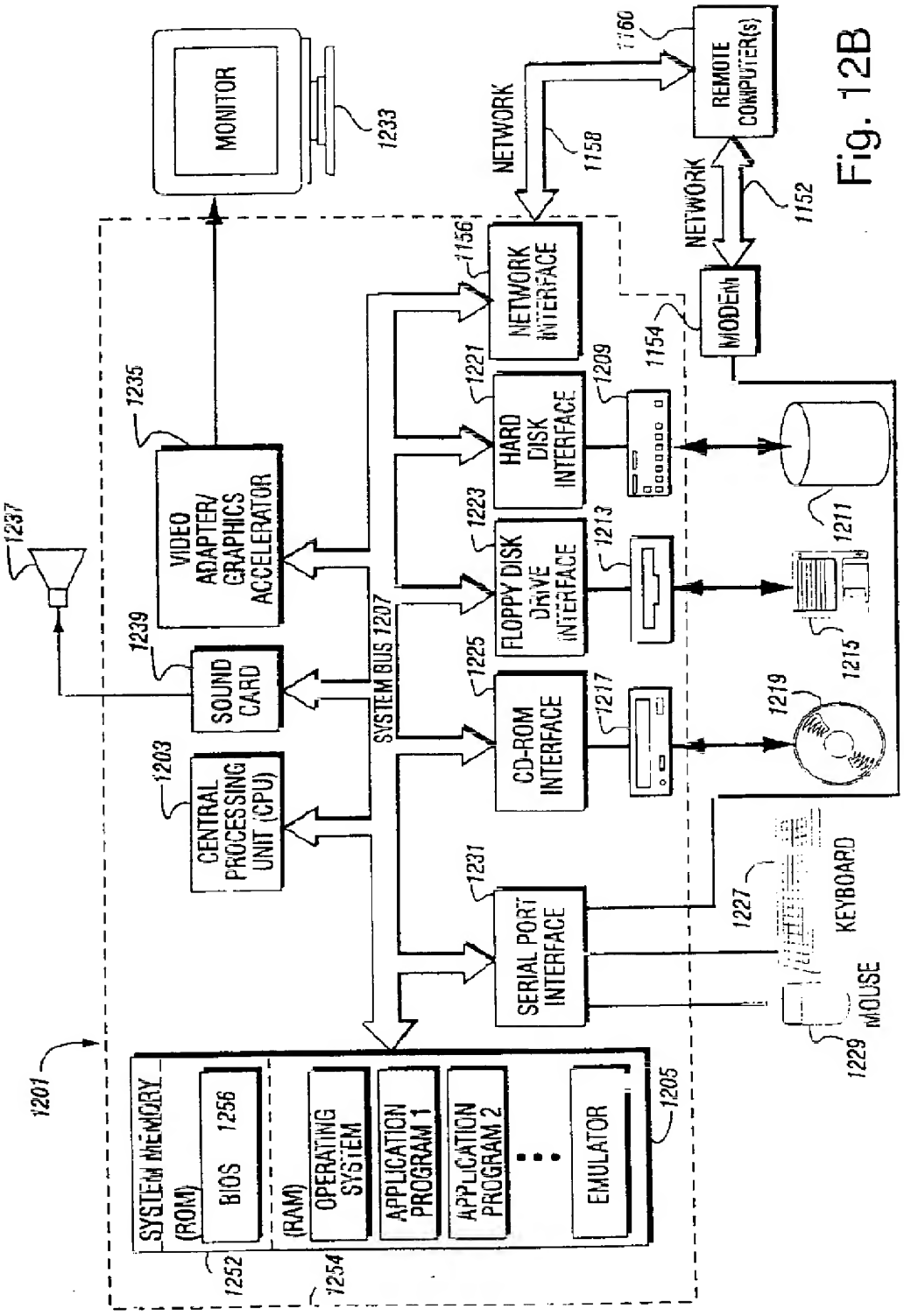

MONITOR
1233
NETWORK
1158
1160
REMOTE COMPUTER(s)
NETWORK
1152
1156
NETWORK INTERFACE
MODEM
1154
1221
HARD DISK INTERFACE
1209
1235
VIDEO ADAPTER/ GRAPHICS ACCELERATOR
1237
1223
FLOPPY DISK Drive INTERFACE
1213
1211
1239
SOUND CARD
SYSTEM BUS 1207
1225
CD-ROM INTERFACE
1217
1219
1215
1203
CENTRAL PROCESSING UNIT (CPU)
1231
SERIAL PORT INTERFACE
1227
KEYBOARD
MOUSE
1229
1201
SYSTEM MEMORY
(ROM)
BIOS 1256
(RAM)
OPERATING SYSTEM
APPLICATION PROGRAM 1
APPLICATION PROGRAM 2
EMULATOR
1205
1252
1254

Fig. 12B

Abstract Of The Disclosure

A graphics system including a custom graphics and audio processor produces exciting 2D and 3D graphics and surround sound. The system includes a graphics and audio processor including a 3D graphics pipeline and an audio digital signal processor. Techniques for efficiently buffering graphics data between a producer and a consumer within a low cost graphics systems such as a 3D home video game overcome the problem that a small-sized FIFO buffer in the graphics hardware may not adequately load balance a producer and consumer – causing the producer to stall when the consumer renders bit primitives. One aspect of the invention solves this invention by allocating part of main memory to provide a variable number of variable sized graphics commands buffers. Applications can specify the number of buffers and the size of each. All writes to the graphics FIFO can be routed a buffer in main memory. The producer and consumer independently maintain their own read and write pointers, decoupling the producer from the consumer. The consumer does not write to the buffer, but uses its write pointer to keep track of data valid positions within the buffer. The producer can write a read command to a buffer that directs the consumer to read a string of graphics commands (e.g., display list) stored elsewhere in the memory, and to subsequently return to reading the rest of the buffer. Display lists can be created by simply writing a command that redirects the output of the producer to a display list buffer.